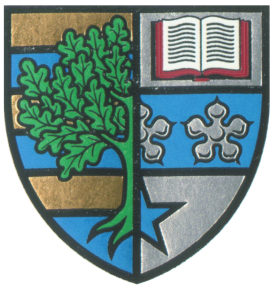
Ubiquitous Integration and Temporal Synchronisation (UbiITS) Framework – A solution for building complex multimodal data capture and interactive systems

Aparajithan Sivanathan

Submitted for the degree of Doctor of Philosophy



Heriot-Watt University

School of Engineering and Physical Sciences

April 2014

Supervisors:

Dr. Theodore Lim

Prof. James Ritchie

Abstract

Contemporary Data Capture and Interactive Systems (DCIS) systems are tied in with various technical complexities such as multimodal data types, diverse hardware and software components, time synchronisation issues and distributed deployment configurations. Building these systems is inherently difficult and requires addressing of these complexities before the intended and purposeful functionalities can be attained. The technical issues are often common and similar among diverse applications.

This thesis presents the Ubiquitous Integration and Temporal Synchronisation (UbiITS) framework, a generic solution to address the technical complexities in building DCISs. The proposed solution is an abstract software framework that can be extended and customised to any application requirements. UbiITS includes all fundamental software components, techniques, system level layer abstractions and reference architecture as a collection to enable the systematic construction of complex DCISs.

This work details four case studies to showcase the versatility and extensibility of UbiITS framework's functionalities and demonstrate how it was employed to successfully solve a range of technical requirements. In each case UbiITS operated as the core element of each application. Additionally, these case studies are novel systems by themselves in each of their domains. Longstanding technical issues such as flexibly integrating and interoperating multimodal tools, precise time synchronisation, etc., were resolved in each application by employing UbiITS. The framework enabled establishing a functional system infrastructure in these cases, essentially opening up new lines of research in each discipline where these research approaches would not have been possible without the infrastructure provided by the framework. The thesis further presents a sample implementation of the framework on a device firmware exhibiting its capability to be directly implemented on a hardware platform. Summary metrics are also produced to establish the complexity, reusability, extendibility, implementation and maintainability characteristics of the framework.

Dedication

Thousands of Tamils in North-East Sri Lanka have been killed during the bloody ethnic conflict. This work is dedicated to those who sacrificed their lives in the war for the community with a communal dream.

Acknowledgements

- For my family, Sivanathan, Sasikala, Arunan, Abhayan and Dhanya:

My parents have been the primary inspiration for undertaking a doctoral degree. They have spent lot of their life and earnings for my studies.

- For my supervisors Dr Theodore Lim and Professor James Ritchie:

Theo and Jim have chosen and offered me with this research opportunity. They were continually encouraging and helping me to shape my interests and ideas. I have learned a lot from them about the professionalism, being open to ideas and ultimately how to perform a good research.

- For the industrial collaborators, BAE systems, Selex, Rolls-Royce, Renishaw and Skanska:

Industrial collaborators have demonstrated their interest involvement and provided feedback at various stages of this work.

- For the funders:

The work was funded primarily by the Engineering and Physical Research Council (EPSRC grants EP/F02553X/1, 114433 and 113946) and the Heriot-Watt University Innovative Manufacturing Research Centre.

- For my colleagues, Raymond, Craig, Zoe, Ying and Matthew:

Colleagues spared their time with me on both the happy and boring parts of the PhD. They advised, assisted and joined me in experiments, testing systems, gathering data and when writing the thesis.

- For the country of The United Kingdom:

This country has generously offered me all the necessary rights, some of which were even refused in my country of origin.

It would not have been possible to read a doctoral degree without the help and support from all of you; I would like to express my sincere appreciation and gratitude.

Finally, a **'Big Thanks'** to all of you.

ACADEMIC REGISTRY

Research Thesis Submission

Name:	APARAJITHAN SIVANATHAN		
School/PGI:	School of Engineering and Physical Sciences		
Version: (i.e. First, Resubmission, Final)	Final	Degree Sought (Award **and** Subject area)	Doctor of Philosophy (Mechanical Engineering)

Declaration

In accordance with the appropriate regulations I hereby submit my thesis and I declare that:

1) the thesis embodies the results of my own work and has been composed by myself
2) where appropriate, I have made acknowledgement of the work of others and have made reference to work carried out in collaboration with other persons
3) the thesis is the correct version of the thesis for submission and is the same version as any electronic versions submitted*.
4) my thesis for the award referred to, deposited in the Heriot-Watt University Library, should be made available for loan or photocopying and be available via the Institutional Repository, subject to such conditions as the Librarian may require
5) I understand that as a student of the University I am required to abide by the Regulations of the University and to conform to its discipline.

* Please note that it is the responsibility of the candidate to ensure that the correct version of the thesis is submitted.

Signature of Candidate:		Date:	21/07/2014

Submission

Submitted By (name in capitals):	APARAJITHAN SIVANATHAN
Signature of Individual Submitting:	
Date Submitted:	21/07/2014

For Completion in the Student Service Centre (SSC)

Received in the SSC by (name in capitals):			
Method of Submission (Handed in to SSC; posted through internal/external mail):			
E-thesis Submitted (**mandatory for final theses**)			
Signature:		Date:	

Table of Contents

Chapter 1. Introduction 1

1.1. Thesis contributions 4

1.2. Hypothesis 5

1.3. Thesis structure 6

Chapter 2. Literature Review 7

2.1. Examples of data capture and interaction 7

2.2. Multimodal systems 9

2.3. On-board computing power 10

2.4. Connecting devices and software systems 10

2.5. Ubiquitous components 11

2.6. Event driven architecture 12

2.7. Standardising interfaces 12

2.8. The synchronisation problem 13

2.8.1. Causes of synchronisation issues 14

2.8.2. Network time synchronisation protocols 14

2.8.3. Latency issues 15

2.8.4. Time synchronisation for data streams and events 17

2.8.5. Online data access and interaction 18

2.9. Computational load issues 19

2.10. Current state of the art tools 20

2.10.1. Multimodal capture software packages 20

2.10.2. Architectures and frameworks 21

2.10.3. Synthesis matrix – Summary of existing solutions 24

2.11. Summary 26

Chapter 3. Methodology 27

3.1. Rationale for a software framework 27

3.2. Methodology of developing a framework 27

3.3. Domain analysis 29

3.4. Requirements derived from the domain analysis 29

3.5. Absorbing ubiquitous concepts 30

3.6. Requirements analysis 31

3.7. Generalisation and abstraction 31

3.8. The framework and reference architecture 31

3.9. Case studies – Implementation examples 32

3.10. Verification and testing of Components 33
3.10.1. Verifying components and functionality 33
3.10.2. Framework metrics 33
3.11. Validation 33
3.12. Presentation of the framework 34
3.13. Summary 34
Chapter 4. Ubiquitous Integration and Temporal Synchronisation Framework 36
4.1. Fundamental use of UbiITS 36
4.2. Reference architecture, layers and components 36
4.3. Framework components - UbiITS toolkit 38
4.3.1. Programming concepts 38
4.3.2. Device abstraction and unified interface 39
4.3.3. State variables for devices 40
4.3.4. Event and stream data types 41
4.3.5. Data flow and real-time data access 42
4.3.6. Asynchronous FIFO buffers for multimodal data transfer 43
4.4. Parallel execution and deployment configurations 45
4.4.1. Standalone deployment 46
4.4.2. Asynchronous function calls and messaging service 47
4.4.3. Distributed deployment 48
4.5. Event driven system – Asynchronous and distributed events 49
4.6. Synchronisation techniques 50
4.6.1. Timestamps 51
4.6.2. Clock synchronisation 51
4.6.3. Latencies in data channels 52
4.6.4. Automatic latency measurement 54
4.6.5. Overall interaction latency 57
4.6.6. Runtime monitoring of sampling 57
4.6.7. Automatic forced syncing intervals 58
4.6.8. Post capture synchronisation 59
4.7. Summary 59
Chapter 5. The application of ubiquitous multimodal synchronous data capture in CAD 61
5.1. CAD - the case for unobtrusive ubiquitous design data capture 62
5.2. Capturing user activity 64
5.3. Requirements for a ubiquitous framework 66

5.4. Experimental setup 66
5.5. Employing UbiITS framework to solve technical problems 68
5.5.1. Bandwidth and computational load 68
5.5.2. Online data exchange 69
5.5.3. Online viewing of captured data 69
5.5.4. Handling CAD system events 70
5.5.5. Integrating multimodal capture tools devices with a centralised control panel 70
5.5.6. Interconnection of distributed devices 71
5.5.7. Integration of ubiquitous devices in the system 71
5.5.8. Data stream synchronisation 73
5.5.9. Handling latency and jitter in data channels 73
5.6. Bio-physical signals synchronised with CAD Events 75
5.7. Design activity representations 76
5.8. Outcomes of the ubiquitous meta data capture 80
5.9. Discussion – Reflections and opportunities 80
5.9.1. Ubiquitous metadata capture environment 81
5.9.2. Interaction and Synchronisation 82
5.9.3. Handling vast quantities of metadata 82
5.9.4. Integrating metadata capture in existing and future CAD solutions 83
5.9.5. Feeling of being watched 84
5.10. Conclusion 85
Chapter 6. A Virtual Aided Design Engineering Review (VADER) system with searchable content: Knowledge engineering via real-time multimodal recording 86
6.1. Current state of knowledge capture in industries 87
6.2. The Virtual Aided Design Engineering Review (VADER) system 89
6.3. Process and formal design record linked knowledge capture architecture 90
6.4. Application of multimodal data capture and interaction framework - UbiITS. 91
6.5. System schematic 93
6.6. CAD model view and interactions 93
6.7. Review interface and multimodal interactions 94
6.8. Distributed web Interface with speech recognition 96
6.9. Timeline, data search and retrieval 97
6.10. Automatic review report creation 99
6.11. Bidirectional linkage to design data and temporal multimodal data 101
6.12. Node - link organisation for design and meta data 102
6.13. Usability and functionality evaluation 103

6.14. Discussion – reflections and opportunities 106
6.14.1. Architecture extendibility – From design review to product life cycle 106
6.14.2. Flexibility over multimodal capture – provided by UbiITS 107
6.14.3. Data usage policies and anonymity 108
6.15. Conclusion and future work 108
Chapter 7. Temporal multimodal data synchronisation for the analysis of a game driving task using EEG 110
7.1. Driving Task and game elements 111
7.2. Need for the temporal synchronisation between in-game data and external data 111
7.3. Game - driving task and psychophysiological analysis 112
7.4. Data capture setup: temporally synchronised driving game logging and psychophysiological signals 113
7.4.1. UbiITS implemented data capture setup 113
7.4.2. Jitter on the data streams 115
7.4.3. Asynchrony of EEG and telemetry streams 115
7.4.4. Using a reference device to calibrate stream offsets 115
7.4.5. Evidence of synchronisation of driving activity and EEG 117
7.5. Brain activity and driving patterns 118
7.5.1. Experimental setup 118
7.5.2. EEG processing 120
7.5.3. Artefacts and noises 120
7.5.4. Interpreting the 'noisy' EEG using synchronised telemetry 121
7.5.5. Potential temporal correlations 122
7.6. Reflections and opportunities 125
7.7. Conclusion 125
Chapter 8. Mirror Game - A Neurofeedback protocol using a sophisticated interactive video game to improve social responsiveness in children with ASD 127
8.1. Background information 128
8.2. Interactive game and real-time feedback 128
8.3. Real-time closed loop feedback and synchronisation 129
8.4. Closed-loop, low latency connection of the user space with computational space 131
8.5. Conclusion and summary 134
Chapter 9. Sensor prototype and hardware implementations 135
9.1. UbiITS built-in hardware 135
9.1.1. UbiITS built-in embedded sensor 135
9.1.2. Hardware architecture 136

9.1.3. Interrupt driven firmware architecture 137
9.1.4. Real-time task scheduling 137
9.1.5. Mutual exclusiveness – safe data access 139
9.1.6. Remote host 139
9.2. Custom calibration device 139
9.2.1. Firmware architecture and latency measurements 139
9.2.2. Sample latency measurement – EEG device 140
9.2.3. Automatic latency measurement module 141
9.2.4. Latency estimation over multiple trials 143
9.3. Summary of hardware implementation 144
Chapter 10. Framework metrics and discussion 146
10.1. Evolution of the framework in parallel with case studies 146
10.2. Software implementation metrics from the case study applications 147
10.2.1. Implementation platforms and portability 147
10.2.2. Framework components usability metrics 149
10.2.3. Cyclomatic complexity and maintainability 151
10.2.4. Supported devices and data types and ubiquity 152
10.3. Functional characteristics 154
10.4. Comparison with the literature 154
10.4.2. Functional novelties 157
10.4.3. Exportability and generic framework 157
Chapter 11. Conclusions and future directions 160
11.1. Future refinements and open ended evolution 161
References 163
Appendixes 187

List of Abbreviations

ADC	Analogue to Digital Converter
API	Application Programming Interface
ASD	Autism Spectrum Disorder
BCI	Brain–Computer Interface
BIM	Building Information Model
CAD	Computer-Aided Design
CLI	Common Language Infrastructure
COM	Component Object Model
CPS	Cyber Physical Systems
CPU	Central Processing Unit
DC	Direct Current
DCIS	Data Capture and Interactive Systems
DLL	Dynamic-Link Library
DMA	Direct Memory Access
DOF	Degrees Of Freedom
DRed	Design Rationale Editor
DVI	Digital Visual Interface
ECG	Electroencephalogram
EDA	Electrodermal Activity
EDA	Event Driven Architecture
EEG	Electroencephalogram
EKG	Electrocardiography
EMG	Electromyogram
EOG	Electrooculography
ERP	Event-Eelated Potential
F1	Formula 1®
FIFO	First In, First Out
GPGPU	General-Purpose computing on Graphics Processing Units
GPS	Global Positioning System
GPU	Graphics Processing Unit
GSR	Galvanic Skin Response
GUI	Graphical User Interface
HANS	Head and Neck Support
HCI	Human–computer interaction
HD	High-Definition
HDMI	High-Definition Multimedia Interface
HTTP	Hypertext Transfer Protocol
ID	Identification
IDEF	Integrated computer-aided manufacturing DEFinitions
IMU	Inertial Measurement Unit
IO	Input/Output
ISR	Interrupt Service Routine
LED	Light-Emitting Diode
MCU	Microcontroller Unit
MFC	Microsoft Foundation Class Library
MIDI	Musical Instrument Digital Interface
MNS	Mirror Neuron System
NFT	Neurofeedback training
OS	Operating System
OWL	Object Windows Library
PAL	Phase Alternating Line
PC	Personal Computer
PLM	Product Lifecycle Management
PSL	Process Specification Language
QT	QT Project
RAID	Redundant Array of Inexpensive Disks
RB7	Red Bull RB7
RBS	Reference Broadcast Synchronization
RTOS	Real-Time Operating System
RTS/CTS	Request to Send / Clear to Send
RTT	Round Trip Time
SDK	Software Development Kit
SPSC	Single-Producer, Single-Consumer
SPMC	Single-Producer, Multi-Consumer
STL	STereoLithography
SUS	System Usability Scale
TCP	Transmission Control Protocol
UART	Universal Asynchronous Receiver/Transmitter
UDP	User Datagram Protocol
UI	User Interface
USART	Universal Synchronous/Asynchronous Receiver/Transmitter
USB	Universal Serial Bus
VADER	Virtual Aided Design Engineering Review
VB	Visual Basic
VBA	Visual Basic for Applications
VGA	Video Graphics Array
VR	Virtual Reality
VTK	Visualization ToolKit
XML	Extensible Markup Language

List of Publications

Sivanathan, Aparajithan, Theodore Lim, James Ritchie, Raymond Sung, Zoe Kosmadoudi, and Ying Liu. "The Application of Ubiquitous Multimodal Synchronous Data Capture in CAD." *Computer-Aided Design*. November 1, 2013. doi:10.1016/j.cad.2013.10.001.

A. Sivanathan, T. Lim, S. Louchart, and J. Ritchie, "Temporal multimodal data synchronisation for the analysis of a game driving task using EEG," *Entertainment Computing*, Mar. 2014. doi:10.1016/j.entcom.2014.03.004

Sivanathan, Aparajithan, Theodore Lim, Sandy Louchart, and James Ritchie. "Temporal Synchronisation of Data Logging in Racing Gameplay." Procedia Computer Science 15 (2012): 103–110. doi:10.1016/j.procs.2012.10.062. **– Awarded as a best paper**

Aparajithan Sivanathan, James Ritchie, Theodore Lim, and Raymond Sung. "A Virtual Aided Design Engineering Review (VADER) System with Searchable Content: Knowledge Engineering via Real-time Multimodal Recording" (n.d.). – *Submitted Article to Journal of Computing and Information Science in Engineering*

Co-authored publications

R. C. W. Sung, J. M. Ritchie, T. Lim, A. Sivanathan, and M. J. Chantler, "The evaluation of a Virtual-Aided Design Engineering Review (VADER) system for automated knowledge capture and reuse," in *Proceedings of the ASME 2013 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference*, Portland, Oregon, USA, 2013. **– Awarded as the best paper in ASME Virtual Environments & Systems (VES)**

Hislop, Matthew, Aparajithan Sivanathan, Theodore Lim, James M Ritchie, Sandy Louchart, and Gnanathusharan Rajendran. "Beyond Simulators: Using F1 Games to Predict Driver Performance, Learning and Potential." (n.d.). – *Accepted Article for Lecture Notes in Computer Science*

J. Ritchie, T. Lim, R. Sung, A. Sivanathan, C. Fletcher, Germanico Gonzalez, Hugo Medellin, Ying Liu, and Zoe Kosmadoudi, "VES: Knowledge Capture in Virtual Reality and Beyond," in *Advances in Computers And Information In Engineering Research*, ASME, 2014.

E. V. C. Friedrich, N. Suttie, A. Sivanathan, T. Lim, S. Louchart, and J. A. Pineda, "Brain–computer interface game applications for combined neurofeedback and biofeedback treatment for children on the autism spectrum," *Front. Neuroeng.*, vol. 7, p. 21, 2014. doi: 10.3389/fneng.2014.00021

Elisabeth V. C. Friedrich, Neil Suttie, Aparajithan Sivanathan, Theodore Lim, Sandy Louchart, and Jaime A. Pineda, "A Neurofeedback protocol using a sophisticated interactive video game to improve social responsiveness in children with ASD," in *Society for Neuroscience, 2013*, New Orleans, LA, 2013, p. 181.15/AAA24.

Liu, Y., J.M. Ritchie, T. Lim, Z. Kosmadoudi, A. Sivanathan, and R.C.W. Sung. "A Fuzzy Psycho-Physiological Approach to Enable the Understanding of an Engineer's Affect Status during CAD Activities." *Computer-Aided Design*. Accessed November 19, 2013. doi:10.1016/j.cad.2013.10.007.

Patents

Ubiquitous Integration and Synchronisation Framework, *Patent application process initiated and it is under review by the Heriot-Watt intellectual properties team at the moment*

Chapter 1. Introduction

Modern information systems are smarter than ever before, possessing more awareness and intelligence about the problem they deal with and closely interacting with the environment in which they are employed. These contemporary systems capture information from various sources and perform well-timed interactions with the environment in numerous ways. For instance data acquisition systems are widespread, varying from life support medical data systems to unattended weather monitoring stations, from human activity logging systems to industrial process monitoring systems, from passive systems merely logging measured physical parameters to highly active systems responding to changes in real-time. These systems often employ various multimodal and heterogeneous tools. The underlying technical challenges in integrating and interoperating these diverse components motivate this work.

Multimodal systems and heterogeneous tools

A host of different parameters can be captured through these systems ranging from physical (e.g. temperature, acceleration, audio, video, and motion) to pyscho-neuro-physiological to name but a few. Systems are designed to handle a single data type or a mixture of multiple data types. Single modal systems are comparatively simple and only deal with one type of data whereas multiple modalities are used in systems that deal with complex tasks, involving data acquisition from multiple dissimilar sources. As a consequence, the use of heterogeneous tools naturally becomes inevitable in multimodal systems that capture both information from the environment and performing interactions back to the environment.

Real-time systems and offline analysis

In an ideal situation information systems should be able to extract all the necessary information from the captured parameters in real-time. However, it is not always feasible to carry out runtime processing; therefore offline analysis is performed in such situations where no pre-known algorithms are available for extracting information in real-time or technical limitations in performing real-time operations (e.g. insufficient computational power, memory, bandwidth, etc.). Offline processing is often used to conduct exploratory analysis, especially where human knowledge and/or intervention is required for analysing the data. The exploratory analysis is performed on the data where the outcome of the analysis is not pre-determined but one can experiment with different strategies and act upon according to the outcome of trials. Therefore large amount of data is collected for an offline analysis and future research more often than not.

Interactive systems

Beyond the task of data acquisition, advanced information systems are sometimes expected to perform timely interactions (e.g. physical, virtual, multimedia, etc.) with an environment, i.e. inducing a change in the environment or modifying the system itself. Elson et al [1] states that the passive behaviour of a sensor node makes it possible to observe the environment but not to change it or the relationship to it; in contrast, an active system when it detects an action may adapt its characteristics to suit. Interactive systems are often used when the capture environment involves active elements, e.g. a living element is present, usually a human-in - loop. Here, the information system reacts to the changes in the environment and induces or influences changes into the environment or vice versa. Basically the interactive information system becomes part of the living environment, blending to form a combined system. Real-time information processing and computations need to be performed in order to adapt system characteristics, provide closed loop feedback and execute timely interactions. Consequently this creates the need for tight coupling across the devices, captured data and information processing tools, in such a way computational algorithms can actively respond to the captured data, perform actuations, control devices and change system characteristics.

Flexible coupling of tools

Components (i.e. hardware/software) used to capture data, process data and perform interactions can be attached to a system either in a fixed or a flexible manner. For instance components can be said to be fixed when they are always available in a same configuration and unchanged from the original design. Alternatively, components can be flexibly attached allowing them to be swapped, go offline or become online, ultimately permitting significant alterations in runtime. Flexible systems are more realistic but difficult to design and interoperate with diverse devices compared to fixed systems because of the different capabilities, interfaces and unpredicted accessibility of devices. They require dynamic runtime behaviour and therefore it is obligatory to have standardised protocols regulating the access to devices and data.

Lack of standardised protocols

Although DCISs come in various configurations and are employed in variety of scenarios, they share many underlying common technical challenges. There are no standardised and scientifically justified protocols are available for addressing the above challenges. Typically DCIS are built in a bespoke manner; consequently uniform and reusable techniques are not available. Various studies have substantiated the fundamental necessity to establish the requirements for common and systematic ways of solving the technical issues that can be shared among the design of complex multimodal systems.

The editors' notes in the proceedings of the "*ACM / SIGIR 2009 workshop on Understanding the User Logging and interpreting user interactions in information search and retrieval*" states the following [2]:

"… More advanced and more implicit interaction data logging becomes more and more popular, e.g. based on eye tracking, skin conductance, and EEG. During the workshop, we identified common needs and problems with respect to logging interaction data. They reached from extracting the focused data from different software applications to merging interaction data streams from different sources. Here, we clearly see a need for a common basis of tools and frameworks shared within the community so that individual researchers don't have to re-invent the wheel over and over again." [2]

Cyber Physical Systems (CPS), an emerging discipline, requires coupling of the computational space with the physical space for intelligent interactions with the use of multimodal hardware software components. The following statement has been made by Baheti and Gill [3] summarises the current problems encountered in building CPS:

"The design and implementation of networked control systems pose several challenges related to time-and event-driven computing, software, variable time delays, failures, reconfiguration, and distributed decision support systems. Protocol design for real-time quality-of-service guarantees over wireless networks, tradeoffs between control law design and real-time-implementation complexity, bridging the gap between continuous and discrete-time systems, and robustness of large systems are some of the challenges for CPS research. Frameworks, algorithms, methods, and tools are needed to satisfy the high reliability and security requirements for heterogeneous cooperating components that interact through a complex, coupled physical environment operating over many spatial and temporal scales." [3]

This clearly indicates that standardised abstractions, architectures and frameworks that permit modular design and development are required to build flexible, extendable robust and predictable systems with heterogeneous tools and modalities [3], [4]. Hollan and Hutchins [5] demonstrated various exploitations of commodity digital multimodal data acquisition tools that enabled them to create smarter systems.

Technical challenges

Common technical challenges encountered in building complex data capture and interactive systems (DCIS) are summarised below:

- Heterogeneous components have dissimilar interfacing techniques. Integrating and interoperating these tools flexibly but robustly is difficult.

- Components in a complex DCIS run asynchronously on detached platforms, i.e. hardware or software processes. Integrating these isolated components and operating them as a system possesses various difficulties. Time synchronisation, multimodal data transport and real-time data access are examples to name a few.

- Computational challenges are associated with coupling real-time asynchronous data flow, information processing algorithms and physical devices, e.g. safe hardware control, thread safety, etc. These issues are needed to be handled both in system level (system software, higher level) and component level (firmware/low level).

In contrast to solving these issues in isolated domains these technical challenges need to be addressed in a generic and standardised manner so as to enhance exportability across a wide variety of disciplines [2]–[4]. Software frameworks and architectures are intended to promote reusability, encapsulate predefined tools, techniques, and a structure to solve a particular problem. The following generic definitions are derived based on various definitions provided in previous studies.

Software Frameworks

A framework is a collection of collaborating abstract and concrete components designed for maximum reuse [6], [7]. They capture both patterns and techniques that implement common requirements and a design of a specific application domain [8]. Generally frameworks are semi-complete applications that embody an abstract design for solutions to a family of related problems [9].

Software Architectures

An architecture is a high level structure of software encompassing the set of significant decisions about the overall the organisation of a system and its components, structural elements and their interrelationships [10]–[12]. While architecture is focused on organizing components to support specific functionality, it also provides a solid foundation for a complex software system.

1.1. Thesis contributions

The major contributions of this thesis are as follows.

- ✓ It presents a software framework as a generic solution for addressing the technical challenges in building complex DCIS infrastructures. The framework is presented in an abstract and generic form so that it is robust, can be adopted and extended effortlessly in prospective applications. A reference architecture highlighting the standard usage pattern is presented along with the framework.

- ✓ It demonstrates and validates the framework's applicability and capability to handle four different application scenarios through case studies. These case studies are real-world problems and not synthetically fabricated to support the framework development. Each case study stands alone and is a novel contribution in its application domain.

- ✓ Contributions demonstrated by the case studies are twofold. Firstly, within the context of each case study application, how the generic framework concept has been applied in the domain, its functionality as a core element to solve the technical issues and outcomes provided by employing the framework. As a result, the success and value recognition of individual case studies can be indirectly counted as credits for the framework. Secondly, from the framework perspective, all the case studies are examples of how the framework can be employed. Each of these examples can be perceived as indications of the framework's conceptual validity, flexibility, adaptability and technical correctness.

- ✓ Importantly, the work exhibits a clear and reproducible methodology for systematically developing a framework.

1.2. Hypothesis

The research hypothesis is:

Building a framework for integrating multimodal data capture, interaction tools and time synchronisation will standardise system design and therefore enhance design reuse and interoperability.

This hypothesis is supported by the following objectives:

1. Review the body of literature and understand domain of multimodal device integration and synchronisation for DCIS and related state of the art technicalities and identify the gaps in the knowledge.

2. Design an abstract software framework that simplifies the design of a complex DCIS architecture.

3. Study and understand application domains and then apply the framework to satisfy the technical necessitates of the DCIS. Extend and refine the framework to satisfy various application scenarios. Demonstrate the framework through application case studies to establish evaluation metrics on the framework.

4. Highlight various novel outcomes in each application case study within their own domain, which are otherwise not possible if not for the framework.

5. Showcase the technical readiness level of the framework beyond proof of concept and fundamental research. Present future directions and identify more opportunities and applications which can be made possible by applying the framework.

1.3. Thesis structure

The structure of this thesis is described in the following diagram (Fig.1.1).

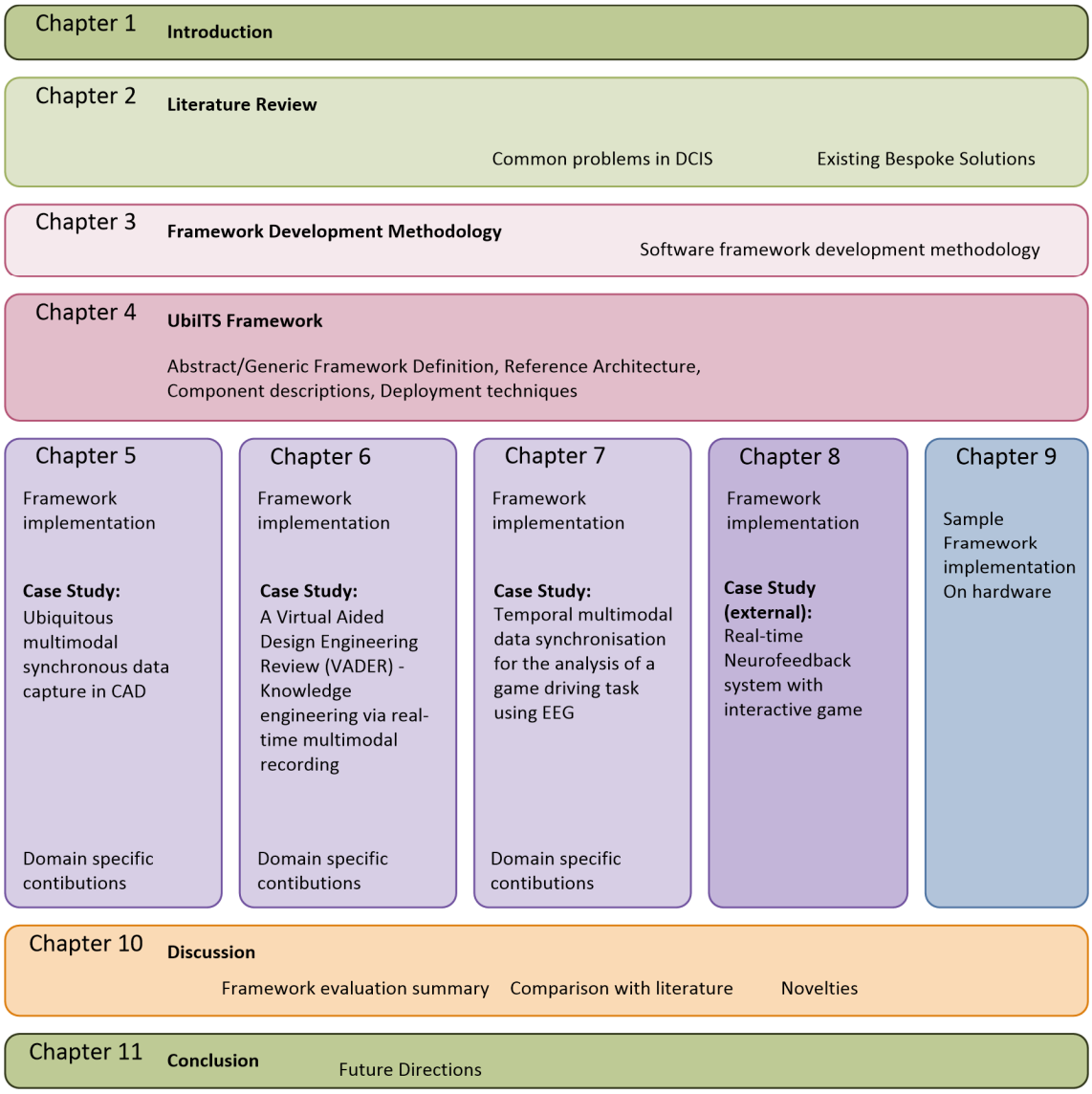


Fig.1.1 Thesis structure

Chapter 2. Literature Review

The literature survey has been performed with the following objectives.

1. To identify what is generally expected from multimodal integrated systems and problems encountered in these systems.
2. To explore how previous applications solved the technical requirements of DCIS, and the effectiveness of their methods.
3. To investigate whether there is a common pattern of requirements in complex DCIS, identify the gaps in the knowledge and if they exist establish a suitable means to address the challenges if nothing is available.

The following materials relevant to this work were included in the literature review.

1. Examples of multimodal DCIS.
2. Specimens of devices and data types used in contemporary DCIS and their interfacing and interoperating techniques.
3. Commercial data capture and interaction software solutions.
4. Fundamental methods of addressing technical problems that arise in multimodal DCIS.
5. Approaches used in building domain specific DCIS, this includes general purpose DCIS, if there are any

2.1. Examples of data capture and interaction

Multimodal data capture systems are employed to monitor and interact with time-varying environments in wide variety of disciplines such as behavioural science, environmental systems and process monitoring systems. A variety of examples of multimodal data capture systems and associated modalities found from previous studies are listed in Table 2.1. The nature of captured data and the way it is used vary among environments, e.g. record keeping, information retrieval, closed loop control, feedback and interaction. Although the primary purpose of each of these applications is different, they share the common set of underlying technical requirements in building their systems. However, in general these applications do not use sharable protocols or tools for building their systems and consequently became bespoke applications. This is a significant limitation in of these applications, limiting the exportability of their outcomes to other domains.

Application	Modalities
Towards Smart Meeting: Enabling Technologies and a Real-World Application [13]	Audio, breast-shot video, face video, meeting browsing
Integration and synchronization of input modes during multimodal human-computer interaction [14]	Speech, Pointing, Click Interactions
Support For Multitasking and Background Awareness Using Interactive Peripheral Displays[15]	Peripheral Display Interaction, PC interactions
Distributed Meetings: A Meeting Capture and Broadcasting System [16]	360 view capture, white board capture, overview video, directional audio, PC interactions, telephone, sound source localization
Real-time analysis of physiological data and development of alarm algorithms for patient monitoring in the intensive care unit [17]	Multiple physiological signals, clinical annotations, alarm triggers
Multimodal segmentation of lifelog data [18]	Audio, Acceleration, Image, Video, GPS data
Experiences of designing and deploying intelligent sensor nodes to monitor hand-arm vibrations in the field [19]	Hand Arm Vibration, Tool ID, Person ID
Synchronization of data streams in distributed realtime multimodal signal processing environments using commodity hardware [20]	Video, 64 Channel Audio, Active speaker localisation
Virtual assistant: an artificial agent for enhancing content acquisition: how ambient media elicit information from humans [21]	Video, image, hand position, object regions, facial expressions, motion, vocalization
The OpenInterface Framework: A tool for multimodal interaction [22]	Acceleration, proximity, head tracking, GPS data, hand orientation, 3D gestures, pointing, touch, click, speech
Cost-Effective Solution to Synchronised Audio-Visual Data Capture using Multiple Sensors [23]	Multiple Audio, Video streams, sync triggers
Unobtrusive physiological monitoring in an airplane seat [24]	ECG, Pressure, Respiration, EDA, Skin Temperature, Finger Acceleration
CALLAS - Conveying Affectiveness in Leading-edge Living Adaptive Systems [25]	Audio, Video, finger postures, wrist movements, hand movements, acceleration, Gaze, facial expression
METABO - Metabolic management in diabetes for both, patients and specialist [26]	steering wheel angle, the vehicle speed and accelerations and the driver's usage of the gas and brake pedals, ECG, blood pressure, GSR, EMG, respiratory rate
IRIS - Integrating Research in Interactive Storytelling [27]	Voice, Emotion, philological data
Augmented Reality Environment for Dance Learning [28]	video streaming, 3D animation, music, textual description, labanotation
Evaluating a minimally invasive laboratory architecture for recording multimodal conversational data [29]	Depth image, body skeleton, head, eye, gaze tracking, audio, video
Nuance: A Software Tool for Capturing Synchronous Data Streams From Multimodal Musical Systems [30]	Audio, Serial port data, Open Sound Control/External streams, MIDI
Predicting User Tasks: I Know What You're Doing! [31]	PC activities, file operations, text operations, phone speech
Knowing the User's Every Move – User Activity Tracking for Website Usability Evaluation and Implicit Interaction [32]	Mouse movements, key presses and scrolling, software activity logging, eye tracking
Interface: assessment of human-computer interaction by monitoring physiological and other data with a time-resolution of only a few seconds [33]	Heart Rate, Skin Conductance - Key Storks and Mouse Clicks, Screen Capture and User observation video, Software logging
A usercentered experiment and logging framework for interactive information retrieval [34]	Software event logging, eye tracking, video, screen capture
Usability Evaluation by Monitoring Physiological and Other Data Simultaneously with a Time-Resolution of Only a Few Seconds [35]	Video, screen capture, keystrokes, mouse clicks physiological signals, phone calls

Table 2.1 Examples for Multimodal Data Capture and Interactive Systems

2.2. Multimodal systems

Reasons for engaging multiple modalities

Our natural and living environments contain multiple modalities, human sensing and interactions in particular, involve multiple modalities, such as vision, audio, touch, speech and movements. Individual sensors are simple to use and capable of measuring a particular parameter. Complex environments though depend on more than one variable. Data captured merely from one sensor may perhaps not be enough to decipher the factual information about the environment but data from more than one sensor could reveal more accurate information [5]. Data from multiple sensors also increase the information detection sensitivity and the capacity of discrimination for a given stimulus. For example, in a driving simulator scenario, an understanding about an accident can be established by knowing the parameters from the car such as steering angle, engine speed and brake force. However a clearer picture of the same event could be drawn when the earlier information is combined with the knowledge about what the driver was doing at the moment just before the accident [36]–[38]. Consequently, complex activities in various circumstances demand knowledge inferred from multiple sensors in order to produce more rich and meaningful information. Integrating knowledge across multiple sensors might represent greater information than that generated from the sum of individual sensors.

Diversity of Data and Devices

Versatile DCISs generally comprise sensing and interaction devices, often from diverse manufacturers. Such devices are wide-ranging in configuration, for example, sensing and actuation devices, hardware devices, hardware-software combination devices, software loggers, built-in data storage facilities, live data streaming facilities, etc. Fundamentally, devices can either be analogue or digital; however, in the digital era, analogue sensors are enclosed in digital interfaces. Although the term ‘devices’ generally represents hardware-based devices, this term is interchangeably used in this work to also refer software-based data logging tools, e.g. user activity logging software.

A prominent classification in devices can be done by the method of or frequency at which the data is being captured. Basically devices either capture data at constant frequency or at sporadic intervals. 44,100 Hz digital audio, 29.97 fps interlaced video, 2 kHz EEG, etc. are few examples of constant frequency capture. Still image frame capture, button presses and mouse clicks are examples of data captured intermittently. Similar to capture devices, interaction devices may perform actuations at fixed frequency, e.g. audio/video rendering or in a discrete manner e.g. pop up messages, beeps, etc. Representing the data being handled in two categories, either in continuous time coded data streams or in events occurring at discrete

times is a common pattern found in various DCIS applications [13], [31], [35]. Stream data generally originates from a continuously changing parameter whereas events are typically used to represent discrete actions taking place in the environment. It is also sensible to capture data in bursts, e.g. reporting mouse movement coordinates only when the mouse hardware detects a movement [39]; however, this is effectively a combination of discrete events and data streams.

2.3. On-board computing power

The new generation of digital devices are often equipped with on-board microcontrollers that enhance the capability of the device. Microcontroller-type devices offer customisable soft properties such as dynamically adjustable gain, frame rate etc. Additionally they enable one device to perform composite functionalities, e.g. devices performing dual roles as both input and output such as multi-touch screens, gaming controllers with feedback. Since they have some - although limited - processing power, they support on-board real-time data processing and therefore deliver processed high-level information rather than just raw data, e.g. Wii-Mote [40], Digital Pen [41]. Signals captured from multiple physical sensors can be combined by the on-board processor to function as a compound sensor device, for example EEG devices with 32/280 Channels [42]–[44], motion capture systems [45]. Availability and affordability of processing power, mass storage and transmission bandwidth maximises the use of commodity microcontroller-based devices that can be connected rapidly and used in a DCIS in order to effortlessly access the ready-made information.

2.4. Connecting devices and software systems

Contemporary hardware-based devices are generally designed to connect with a PC platform, possibly supporting one of the popular OS kernels, e.g. Windows, Linux. Device manufacturers usually provide a custom platform driver that adheres with the platform interface specifications, e.g. human interface devices [46], Windows sensor API [47], Linux device model [48], etc. Further to the device drivers, a device specific API or SDK might be provided which can be used by programmers to design applications accessing device functionalities and data from/to the device [49], [50]. Shared Dynamic Link Libraries, Component Object Model, Pipes, Active X Controls, Custom communication Protocols built on Serial Ports, TCP Sockets and UDP Sockets are examples of fundamental communication objects used by various DCISs (Table 2.1) to link with APIs and software components. Some manufactures or third parties may be able to provide direct interfacing protocol guides if the devices support open interfacing techniques such as TCP, UDP, RS232, etc. [51], [52].

Generally programmers need to access various software functions provided by the device driver or API to control the device and read/write data. For example, functions for initializing the device, releasing the device, starting data capture, stopping data capture and configuring parameters read/write data are typically available control functions in an API. Although the actual tasks performed by control functions are often similar, naming conventions and ways to access these functions vary significantly across different device manufactures [53], [54]. A possible reason is attributed to different connection and protocols, e.g. USB, Ethernet, Wi-Fi, Bluetooth, Serial, Parallel.

It is also common to find hardware-devices operating in conjunction with a software counterpart running on a PC side, e.g. motion tracking systems, eye tracking systems - infrared cameras capture raw image frames while the PC software detects bright markers or eye features in image frames [50], [51]. Physical interaction is not possible without a hardware device, nevertheless on many occasions it is acceptable to consider software as a device. For instance in a browser activity logging system, it is more practical to consider higher-level interaction activities such as webpage navigation, scrolling, window changing, etc. instead of lower level mouse movements and key presses; nevertheless multiple lower level activities as a combination turn into higher level activities. In such cases a logging system can consider the software as a 'device'. In addition to data collection and interaction device nodes, observations gleaned from various application systems reveal that various auxiliary nodes are generally employed in a DCIS, such as real-time processing components and storages devices. These nodes can be software, hardware or combination nodes. For instance, GPU hardware real-time processing technique [55] is a hardware-accelerated computational component that can be used in a real-time DCIS.

2.5. Ubiquitous components

In contrast to connecting various devices to a PC centred platform another approach is to layout devices in a ubiquitous fashion [56], [57]. In this configuration, computational devices are made available throughout the physical environment while making them unobtrusive to the user [58]. Devices in a ubiquitous configuration need to be interconnected, self-governed, cooperating and functioning as an autonomous system [59]. Beyond capturing data samples or performing an actuation there is an inherent necessity for ubiquitous devices to sustain their own computational needs, such as networking, responding to commands, self-priming and maintaining local resources. Although there is no restriction for integrating standard PC platforms as nodes in this configuration, generally devices are expected to exist independently in a ubiquitous system [60]. Consequently it is important to access device functions in a

transparent manner so that other parts of the systems are aware of all the available functionalities and hence interoperate efficiently.

Various components are expected to be flexible so that they can be attached and detached spontaneously in run-time is a generally a desired characteristic of ubiquitous systems [61]. Kindberg and Fox state:

"A component interoperates spontaneously if it interacts with a set of communicating components that can change both identity and functionality over time as its circumstances change. A spontaneously interacting component changes partners during its normal operation, as it moves or as other components enter its environment; it changes partners without needing new software or parameters." [61]

Many studies in the past have emphasized the importance of moving from the conventional PC based computing based paradigm to adopting ubiquitous system concepts, where sensing, interacting and computing nodes are embedded unobtrusively into the ambient environment and within day-to-day activities [62]–[64]

2.6. Event driven architecture

Event driven architecture (EDA) is a commonly used method for building complicated action-reaction-based systems [59], [65]. In this approach events are generated by event-producers based on various conditions such as user activity, state changes and threshold met. Events occurring in the environment may trigger various reactions. Event-consumers process the triggers and react to them in various ways such as controlling a physical parameter in the environment by actuating a device and generating/sending messages etc. They may also generate consecutive events, literally performing a chained, dual role. The principle advantage here is that event producers and event consumers are loosely coupled [65]. This allows the easy swapping of actions and reactions.

For this reason EDA is generally a preferred characteristic for ubiquitous systems, distributed systems and smart environments as well as being used commonly in standard PC-based applications [66]–[70]. EDA is a promising approach for constructing a DCIS where complex interactions are often performed by flexibly coupled components.

2.7. Standardising interfaces

Not only do devices come in various configurations but interfacing techniques also differ greatly among manufacturers. Goudeseune and Kowitz [53] stated: "*Hardware vendors often assume that system integration revolves around their own devices*". This statement is still true

and applies to the majority of multipurpose devices. There is a small number of standards to define uniform interfaces across devices, e.g. Integrated Network Enhanced Telemetry (iNET) [71], Airbus IENA [72]; nevertheless they are not intended to be used beyond highly regulated bespoke disciplines such as aircraft telemetry. There is however no standard established and technically validated to be used in a generic DCIS.

Interoperability among diverse devices, the ability to loosely couple components and transparency in ubiquitous systems stimulates the requirement for standardised component interfaces. Additionally, device interfaces and APIs are targeted for advanced users and therefore require vigilant use by the programmer [73]. Mishandlings often result in scenarios where the devices and software components become unstable, communication interrupted, timing errors [74], data corruption and memory leaks [75] occur. As a result, programming logic gets complicated when integrating multiple devices with diverse configurations. Interfaces and communication protocols are often exclusively built for the particular device.

While it would be ideal to have standardised interfaces defined, trying to standardise proprietary interfaces is an enormous task. There have been a few efforts in the past to define various interface standards tailored to certain application domains, e.g. hardware independence across test and measurement devices in instrumentation systems, virtual PCs, etc. [76]–[78]. These standards neither attract nor are intended for a wide cross-domain usage. Probably defining strict bespoke interfacing strategies would backfire as they are not going to be adopted for a wide use. A promising approach of factoring devices based on their functionalities is proposed in VRPN framework, designed for VR systems [79].

In summary the findings point to the importance of defining a minimal interfacing strategy that is preferably an intermediate, mutual interface separating custom component specific interfaces from the shared interfaces. Coarser interfaces on the other hand may result in sacrificing minute functionalities provided by components; so a balance between rough abstraction and micro control is necessary. Therefore a cohesive interface for accommodating common and essential functionalities and further extendibility for supporting micro control is necessary.

2.8. The synchronisation problem

Synchronisation is a well-known problem and widely studied in distributed sensor networks [80], [81], [1]. In a time varying environment, time phased logs from different sensor locations and recording activities provide information about the global system behaviour. Precise time measurement is necessary to reconstruct the exact sequence of events, particularly where tractability is required for critical event related or chronological, tasked-based operations as in

event related potential (ERP) studies in neuroscience [82]. Accuracy of the information retrieved from the data is skewed if it is not adequately synchronised to an appropriate reference time, preferably an accurate, central or common clock source. Furthermore, when simultaneous data from multiple sensors are involved in deciphering information, temporal alignment of data channels becomes necessary to produce valid information [83]. Unaligned data can result in skewed information, such as missed events, detection of false events or missed cause-and-affect relationships. For example, results generated by multimodal fusion [84] depend on up-to-date readings from multiple sensors and temporal alignment between corresponding data streams. Synchronisation among such sensor groups plays a key role in the status updates emitted by the fusion unit. Causes of this synchronisation problem and the existing methods to solve these problems are reviewed below.

2.8.1. Causes of synchronisation issues

Assorted devices and software component nodes in a DCIS running on isolated platforms possibly with physically separated hardware and system clocking essentially run asynchronously. It is normal to observe transmission delays and jitter-distortions while accessing live data streams [85], [86]. Even though these nodes might be internally capable of sampling signals at sufficiently precise intervals, general purpose OSs and connectivity solutions (USB, TCP) create nondeterministic distortions in the synchronisation of the data streams [28], [86]. Although special and coupled clocking features, e.g. sync signal connection, are available on limited hardware; commodity versatile devices do not often provide advanced clock/stream synchronisation methods. Nevertheless, it is not practical to expect a common synchronisation technique/feature in any ubiquitous data capture environment where diverse and multimodal components are used [53]. Sensor nodes running with the aid of general purpose OS based platforms tend to suffer huge timing issues [87], [74], [88]. The most apparent reason is general purpose, multitasking OSs are not designed to handle precise real-time scheduling and multi-tasking needs [83].

2.8.2. Network time synchronisation protocols

A broad-spectrum of time synchronization schemes are available to synchronise clocks across sensor nodes [89]–[92]. Although these methods are satisfactory for various distributed system deployments, the main problem with this method is that not all the devices come equipped with time synchronisation features [93]. Some hardware does not possess a clock at all and consequently will have no capability to timestamp the data but simply stream the data out. Therefore, it is up to the receiver of the data to cater for synchronisation.

Common synchronisation methods provided by most of the capture devices can be categorised as follows: 1) every sample is time stamped with a local clock inside the sensor device; 2) every sample is time stamped by polling a global clock [53]; 3) no time stamping is performed instead data is streamed at a fixed frequency; the collecting node will decide how to timestamp. 4) data is sampled with triggers provided by the control software or by using an external triggering device.

2.8.3. Latency issues

The data transmission across various locations is an essential aspect of every DCIS. The transmission of data can take a variety of forms, e.g. signal streams, events, messages, commands, - and takes place between various nodes of the system. Various nodes networked generally use either one or a mix of connectivity solutions, e.g. Ethernet, Bluetooth and Zigbee. Apart from the interconnection of components, data may need to be transported inside the component itself, e.g. reading from hardware buffers. Inevitably the data transportation process creates time latencies between the data that is being written/sent (ready to be read) by the origin node and read/received (actually being read) by the destination node. The latencies can vary from few clock cycles to long time durations, depending on various aspects, of the underlying platform, e.g. clocking speed, transmission protocol used, traffic on the data channel, buffers in the data path, packetisation process. [94]–[96]. Example numeric values for these latencies can be found in later chapters (see: Sections 7.4.4, 9.2.4)

Sivrikaya and Yener [81] decompose the sources of the delay in a sensor network as send time, access time, propagation time and receive time. These are correspondingly associated: to 1) creating the message, 2) waiting for a free communication channel to transmit the message, 3) time spent in the propagation and 4) receiving and decoding the message. In addition to the delays incurred in the data path, sample acquisition latencies also influence the synchronisation. Latencies introduced in measurement in a multisensory fusion system are identified by Kaempchen and Dietmayer [80] as measurement acquisition time and pre-processing time latency due to the communication transfer time. An example of a sensor node performing video capture of a live environment is given in the Fig.2.1. Nevertheless, accurate numerical data related to this measurement latency can only be discovered using the manufacturer's test rigs and from design specifications. This data is device/platform depended and difficulty to reproduce; usually these specifications are not available for standard customers.

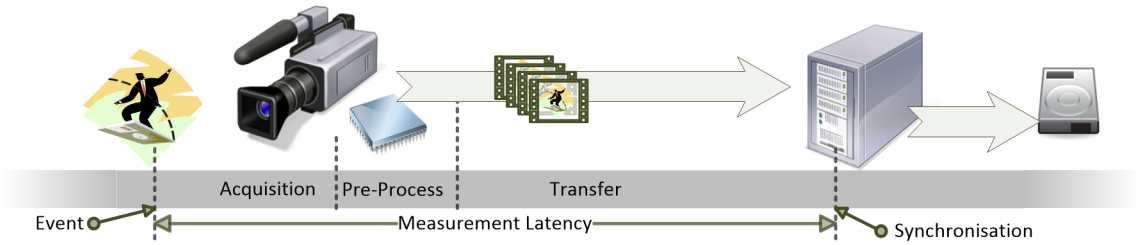


Fig.2.1 Acquisition Latencies. Latencies in involved from acquisition until synchronisation point

Due to the complexities in estimating the factual values, many studies have ignored latencies completely [22], [34], [97]. For instance, "*Sensing architecture for Empathetic Data System*" assumes the latency between the input and reaction is negligible, despite the claim made in the study that it synchronises multimodal data streams.

"*Finally, the delay between user input (explicit and implicit) and presentation should be prompt enough to guarantee a fluent interaction between user and system. Hence, processing and interpretation of the recorded signal must happen in (near) real-time, while also a communication layer is required to allow the different subsystems to exchange messages in an efficient way*." [97]

Although it might be appropriate to make this assumption in this specific circumstance, porting such a solution into other applications could result in unwanted effects. For example it might not be important to consider sub-second latencies for a study analysing chronological ordering of mouse clicks; however, sub-second time skews can create artefacts in a psychophysiological signal analysis that tries to correlate mouse clicks, e.g. in sensorimotor ERP studies [98], [99]. Nevertheless the latency problem is well known aspect in the conventional multimedia discipline (audio-visual presentation), i.e. lip-sync [100], [101]. A comprehensive analysis and comparison of the renowned multimedia inter-stream synchronization approaches can be found in a review by Boronat et al [95].

Although not affecting the dedicated sync cablings by a large amount, general purpose connectivity solutions suffer non-deterministic characteristics [81]. This in turn causes the transmission latencies to be variable rather than constant; this phenomenon is generally denoted as jitter in data channels. Peripherals connected to general purpose OS also create this effect e.g. USB hardware connected through an API in Windows platform. In this case latency fluctuations are caused both by packet delay variation in the data transportation and non-deterministic task scheduling characteristics of the OS [74], [87], [96]. A general solution to deal with the non-deterministic latencies is to introduce a jitter buffer; however this extra buffering also adds up to the overall latency [102].

2.8.4. Time synchronisation for data streams and events

A possibility of achieving synchronisation is running the data capture asynchronously on independent capture devices but inserting markers into the output streams when they receive an event. This method is applicable to devices which do not have clock synchronising features. This is comparable to the post-facto synchronisation approach - Reference Broadcast Synchronization (RBS) suggested in [103]–[105] where a third party beacon reference signal is used to reconcile data streams immediately after an event. Although the markers sent to individual data streams can arrive at different times, inserting markers, similar to RBS, reduces the non-determinism and enables quantifying time offsets. The authors' claim in RBS is that a single broadcast will propagate to all receivers at ideally the same time and hence the propagation error is reduced. Fig.2.2 illustrates measurement latencies and synchronisation offsets assuming the markers are inserted in the streams at the same instant.

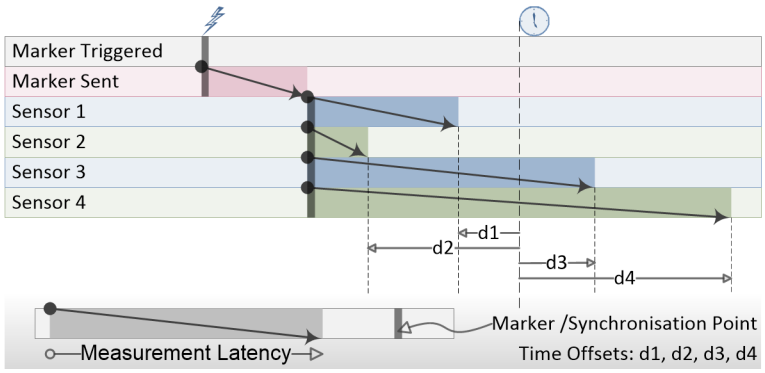


Fig.2.2 Time offsets of streams at an instance of time. This method assumes that markers are inserted in every stream at the same instant

Another approach mostly suitable for post-capture synchronisation is source data based synchronisation which is also used in several studies. This method is based on identifying coherent patterns in the data streams [93], [106]. Sensor clock offsets are calculated using the temporal alignment of the patterns relative to the other sensor streams. Normally a calibration action is used to create a detectable pattern in the sensor data. This technique requires an 'action' that can be executed to inject a data signature pattern in the data stream [93]. The main advantage of source data based synchronisation technique is that it can be used even wherever no specification of latencies in the capture device is available since it relies only in the action-source and the output-data. Finding a suitable action or actions, which can spawn a common signature pattern in multiple sensor nodes is not always practical; for instance, synchronising temperature and audio streams. In some circumstances it is almost impossible to detect a signature pattern especially when there is no prior knowledge available about a

recorded signal, e.g. response signal pattern to a stimulus in EEG signals. In these cases inducing a known artificial pulse can possibly solve the problem.

A naive approach commonly used for synchronisation post capture is to align captured samples based on the beginning and end of the recording. This approach is vulnerable to dynamic characteristic changes in asynchronous data channels, such as clock drift, jitter, etc. An assumption of a constant frequency of sampling and the disregarding of dynamic behaviour of synchronisation are the main problems with this approach. For instance a deviation of ±0.01% Hz in the streaming frequency would result in ±360ms deviation in the duration of an hour. While near ideal systems with dedicated precision clocks and connectivity may not to a large extent be affected, devices running on general-purpose platforms with isolated clocks are affected significantly e.g. Windows task scheduling, wireless connectivity. Another disadvantage is that this approach implicitly requires both asynchronous data streams to be started/stopped simultaneously which is difficult to achieve in practice.

2.8.5. Online data access and interaction

In contrast to capturing and storing the data, instantaneous access to the data being captured is required at various circumstances, examples include data fusion, data visualisation, interaction and feedback to name but a few. In various experimental data logging scenarios online data visuals can possibly indicate signs of quality of data being captured. Therefore they are used to inspect the quality of the recording and identify any faulty sensor nodes which need to be fine-tuned or re-configured. Availability of online data can be used to impact and influence the surroundings and hence initiate state transformations on the environment. For instance, in a smart meeting system [13] real-time browsing of captured information allows participants to take a bird's eye view of the meeting and thereafter the organiser can perform adjustments to the meeting to be more efficient [13], [107]. Similarly systems using virtual assistants [108], [21] discover the current context of the environment and may offer with relevant information and hence demonstrate the online information push. Human in-loop scenarios with bidirectional interactions and feedback are another group of applications demanding access to the online data, e.g. neuro-feedback systems [109].

The conventional practice of multimodal data capture is the all-inclusive capture of multiple sensor streams and the exhaustive searching for information during post processing. The problem with this method is information overload. The excess processing, retrieval and handling of the data comes with extra costs such as processing power, storage, energy and man-hours. Habitual environments are normally a combination of multiple activities. Consequently captured raw data consists of relevant and irrelevant information. The relevancy

is defined by the phenomena of interest whereas the irrelevancy is caused by the addition of information from tangential activities. The data associated with this extraneous information could be identified and removed at the early stages or preferably avoided in the first place. Therefore the raw data consisting of unwanted information is considered to be low-level and takes the form of raw measurements. Ideally it is more efficient to process the raw data in real-time and produce higher level, pertinent information as early as possible. Maintaining the system in an adaptive fashion can potentially be useful in this case; this can be achieved by processing the information in real-time and mobilising the available resources in an on-demand basis. For example, video-recording starts only when a movement is detected or audio is recorded only upon voice detection.

Generally, these examples substantiate the fact that a DCIS needs to be adaptive and dynamic to interact with the environment and reconfigure itself to match its instantaneous needs. Latency issues described in section 2.8.3 create inherent latency problems for real-time interaction and dynamic adaptability. Fig.2.3 shows an example regarding a timing illustration of an interaction performed by fusing information from two sensors, where transportation, jitter and online processing introduce latencies. All these factors influence and cause the overall latency to increase. Interaction latency should be adequately minimised so that it does not create any artificial effects in the natural interaction process, particularly for human-in-loop systems [110]–[112]. The latencies are largely platform specific and it is extremely difficult to find a unified and reproducible value. Indeed, various interaction latencies of an application dealing with sound detection across Android mobile platforms have been listed by a developer community [113].

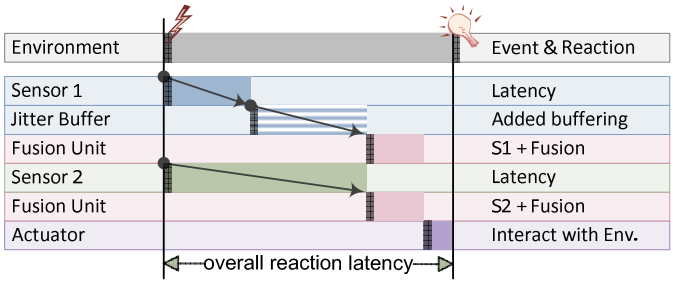


Fig.2.3 Reaction latency

2.9. Computational load issues

Handling computationally expensive tasks, particularly real-time processing components, is well known [34], [114]. Ubiquitous devices are always limited in resources, processing power and memory [59] yet they are generally expected to perform time-critical tasks. Small mobile and embedded devices employ strict timing policies and distribute the load among dedicated

components specialised for certain tasks, e.g. mp3 encoding/decoding chip. On the other hand resource capacities of PC platforms are always increasing. Contemporary computational units, e.g. GPGPU and multicore CPUs utilise parallel threads to deliver superior processing capabilities. As a result, absorbing the concept of concurrent multi-tasking and parallel processing into the inherent design is desirable for DCISs. Conversely, exchanging data and events across concurrent tasks and asynchronously running component nodes require careful programming related considerations, i.e. other nodes determining when a node has finished reacting for an event, prioritising tasks, sharing resources etc. [59], [115]. These issues are generally addressed with multi-threading related primitives e.g. locks, signalling, mutual exclusive access, thread synchronisation, etc.

High data transfer bandwidth requirements are also common in many circumstances, e.g. HD Video, multi-channel, physiological signals, etc. High data rate transfers can peak the bandwidth of the data channel; therefore, it might be necessary to perform online data compression in some circumstances. Serving real-time task requirements together with computationally intensive tasks naturally leads towards mixed-criticality task scheduling needs [116], [117].

2.10. Current state of the art tools

Thus far technical issues related to DCIS were analysed to establish an understanding of the domain. Various studies have tried to address these issues in an isolated manner; apparently there is no single solution to address all them as a collection. Proposed solutions to address these problems collectively are reviewed in the following subsections. Solutions from the literature which address some if not all of the relevant issues are reviewed here.

2.10.1. Multimodal capture software packages

There are various commercial and open source software packages that are designed to accompany data capture devices, e.g. Captive L2100 [118], D-LAB DataStream [119] Bio-Trace+ [120]. Many of these provide extended features to support commonly used data capture components (e.g. video, audio), presentation components (e.g. movies, flash presentations) as well as some triggering options for the purpose of data synchronisation, (e.g. Serial Port Trigger byte). Generally these packages are highly centred towards a particular product or application, for example solutions for physiological signals capture/analysis.

Documentation and specifications about the internal architecture of commercial tools are typically not available. The predominant problem in these products that it is very difficult to use them beyond the purpose they are originally designed for. Secondly a complicated DCIS

cannot be centred on a bespoke software package that tries to extend its support by add-on features like patches. It is important to have a structured architecture that is open to including upcoming devices and data types so that it can function as a systematically organized and flexible system; this is practically impossible with fixed software products. Furthermore, these packages are limited to one/few PC platforms and therefore cannot be employed in ubiquitous or distributed configurations.

2.10.2. Architectures and frameworks

Unlike software packages, architectures and frameworks are intended to be used and customised as per an application's needs. Definitions for architectures and frameworks have been given in Chapter 1. There are only a small number of reusable architectures or frameworks proposed with a view to building DCISs. These are reviewed below.

Social Signal Interpretation (SSI) Framework

Social Signal Interpretation (SSI) Framework [121], [122] is a tool developed specifically for aiding the jump-start development of online emotion recognition systems. This framework was designed with the view to transforming the paradigm of offline emotion recognition into an online process. This SSI integrates of wide range of multimodal components, e.g. audio, video, hand tracking, face tracking and philological signals and it allows piping of the data streams to various online processing and classification modules. The accuracy of such a signal processing tool is depended and the recognition of tasks will be affected by the temporal characteristics of streams supplied to it. Misalignments in the input signal streams can influence the output generated by the tool which fuses the data from multiple signal streams. Therefore the samples from multiple streams need to be indexed with regard to a common reference so that the processed output can also be referenced to the input streams and subsequently related to the source activity. Although this work mentions the synchronisation of multiple channels, detailed information about how, and the granularity of, the synchronisation are not transparently indicated. While a substantial number of multimodal data capture devices were already built-in by the authors, this framework does not define a promising method to integrate future devices and guide external users.

Open Interface (OI) Framework

Open Interface (OI) Framework [22], [123] was developed to support the rapid development of multimodal input interfaces. OI includes a graphical development interface to quickly assemble a multimodal interactive system from predefined modules where these modules can be interconnected using visualised links. Existing components, such as device drivers, interaction

processing modules, multimodal and fusion facilities are all contained within a component repository which is available for a user to build an interactive system. This component collection includes generic and tailored components and is intended to be updated by the authors as stated. Similarly this work does not provide detailed guidelines for users integrating their own set of devices and tools. In essence the importance of generic toolsets and synchronisation has been realised at a conceptual level in OI framework, however sufficient technical details about implementation and extendibility were not indicated or evident.

EyesWeb Extended Multimodal Interaction (EyesWeb XMI) Architecture

EyesWeb Extended multimodal interaction (EyesWeb XMI) architecture [124], [125], [114] was initially developed to map auxiliary modalities, e.g. non-verbal expressions, gestures, body movements with conventional audio visual multimedia data. It later was extended to other data modalities such as physiological data [125]. Similar to OI, this solution is centred on a graphical development environment where devices and tools are represented with blocks. This architecture is composed of prioritised kernel objects, patch objects, customised sub-patches, clocks, interconnection pins and links. While synchronisation issues related multimodal data with commodity hardware and general purpose OS have been taken into account there is no information about how the issues related to latencies in data channels are handled and no quantitative evidence about synchronisation accuracies are provided. The possible reason for discounting these essentials could be that this system is mainly targeted to data streams with low latencies and/or symmetric paths, e.g. audio interfaces. Although the authors have indicated building standalone applications are possible, the main problem with this solution is that presumably, independent external systems without having dependencies to in-house components cannot be developed. A portable architecture should be defined which provides generic components, abstract interfaces and deployment strategies, enabling users to transform and absorb them within their own system with commissioned devices and modalities.

Pure Data (Pd) and 'Patcher Programming Languages'

The idea behind the graphical programming model for controlling dataflows across real-time components was initially originated from the family of 'Patcher Programming Languages' [126]–[128]. Pure Data is a popular and actively maintained tool for interfacing sensors, input devices and MIDI devices for the purpose of generating real-time multimedia content, i.e. sound, video, 2D/3D graphics [128]–[130]. Pure data systems are primarily composed of core, built-in Pd objects, abstractions or reusable patches created in Pd itself and extended externals objects developed in other programming languages. The main target domain of this tool is

multimedia content generation. The fundamental concepts and techniques of data transfer are also applicable to a general purpose DCIS.

User-centred Experiment and Logging Framework for Interactive Information Retrieval

A study by Bierig et al [34] proposed a framework for logging multimodal data for obtaining rich behavioural data in task based experiments. This study highlights few imperative, fundamental characteristic requirements of a data logging system such as loosely coupled devices, scalability of the setup, stability, fail-recover features and computational power. Nevertheless this work only demonstrates one experimental scenario with fixed devices, and therefore does not exhibit sufficient evidence to be considered as a general purpose interaction logging framework.

Tracking Real-Time User Experience (TRUE) Architecture

Tracking Real-Time User Experience (TRUE) Architecture by Microsoft Game Studios [131] is a system for recording data to study user behaviour in a PC software or a gaming console. This architecture looks at streams of data and logs sequences of events along with the timestamp. Captured video is also synchronized with the event timestamps and indexed based on the events. While this demonstrated the efficient retrieval and navigation through the captured, large data sets, complex time synchronisation issues such as latency and jitter in channels are not addressed in this work. This solution does not appear to extend itself towards other data modalities or diverse devices.

Multimodal Data Capture and Analysis of Interaction in Immersive Collaborative Virtual Environments

Steptoe et al [132] presented an architecture for synchronous multimodal data capture specifically in immersive VR systems. The VR system used in this study is focused on capturing multimodal interactions with VR-based, immersive avatar-mediated communications. Technical problems such as multimodal temporal synchronisation, bandwidth issues, and computational power limitations are identified and reported in this work. Unsurprisingly, these issues are common in almost every complex DCIS. The authors have proposed the possibility of using this architecture as a reference-architecture in other VR systems. Nevertheless, this architecture is highly tailored to the specific system arrangement and devices, and therefore falls into the group of bespoke solutions.

VR Development Tools: VR-juggler, VRPN

There are few prevalent frameworks for developing VR applications which address various significant concepts that can be adopted into standard DCIS systems. In spite of everything, VR systems are a specialised form of multimodal interactive systems containing many sharable fundamental technical aspects. The VR-juggler framework [73] was developed to simplify VR application design so that developers do not need to pay attention to every low level technical aspect of IO devices. VRPN [79] is particularly designed for providing device-independent and network transparent interfaces for VR peripherals. Even if these frameworks are not focused on complicated DCIS development, they proposed few conceptual ideas, such as unified and extensible interfaces for heterogeneous tools, runtime flexibility, runtime device control, failure management and multiple simultaneous connections to data channels are adaptable to generic DCISs. In particular, a module for measuring sample delivery latencies was proposed for online performance monitoring and debugging purposes [110].

2.10.3. Synthesis matrix – Summary of existing solutions

Following this review of the most relevant work in this area, Table 2.2 includes a number of other solutions and summarises the functionalities exhibited by each system. It should be noted that the functionalities are indicated as claimed by authors, even though critiques were made previously.

The review on the application examples and system solutions involving multimodal data capture and interaction reveals that the technical problems and gaps are well recognized and widely shared across multiple DCIS systems. The majority of solutions provided in the literature were designed to address a specific or a set of similar applications in a bespoke manner. Although a few of them have tried to disseminate their solutions towards other domains, they have all failed to show sufficient generalisation and completeness. Therefore, no single solution could be endorsed for all diverse DCIS applications. Previous works have identified and proposed a number of promising approaches to solve these technical issues in an isolated manner; however these are not found anywhere in a collective, effective and usable manner for practical system building.

	Temporal Synchronisation	Events	Streams	Real-time Data Accessibility	Real-time Interaction with environment	Data Transport	Unified interface	Device Status/Failure Management	Commodity Hardware /Software, OS	Ubiquitous , Runtime flexibility, reconfiguration	Tool Kit/Modular Building	Graphical development	Generic /Abstract Definitions	Reference Architecture	Trans-domain applicability -Framework	Target Application
Social Signal Interpretation (SSI) Framework [121], [122]	●		●	●							●					Online Emotion Recognition
Open Interface (OI) Framework [22], [123]	●		●		●							●				Multimodal Input Interactions
EyesWeb Extended multimodal interaction (EyesWeb XMI) Architecture [124], [125], [114]	●		●						●			●				Multimodal data mapping on to Audio visuals
Pure Data [128]–[130]			●	●							●	●			●	real-time multimedia content, audio visuals
User-centred Experiment and Logging Framework for Interactive Information Retrieval [34]	●								●					●		behavioural data logging in PC tasks
Tracking Real-Time User Experience (TRUE) Architecture by Microsoft Game Studios [131]	●	●														Game data logging
Multimodal Data Capture and Analysis of Interaction in Immersive Collaborative Virtual Environments [132]									●					●		Multimodal VR interactions capture
VR-Juggler [110], [133], [134]							●	●		●	●		●			VR
VRPN [79]						●	●						●			VR
Automatic Event-based Synchronization of Multimodal Data Streams from Wearable and Ambient Sensors [93]	●		●													Physiological data capture
NIST Data Flow System [20], [96]	●		●	●		●			●							Audio, Visual
Synchronous data collection from diverse hardware [53]	●															Driving activity capture
Bio signals Studio [135]			●	●												Physiological data logging + voice, video
Service-oriented smart home applications: composition, code generation, deployment, and execution [136]								●								Smart user spaces
A Database-Oriented Wrapper for Ubiquitous Data Acquisition/Access Environments [54]							●	●		●						Video, Identification
Finite-State Machine Based Distributed Framework DATA for Intelligent Ambience Systems [137]		●						●								Intelligent ubiquitous systems
W3C - Multimodal Architecture and Interfaces [138]		●			●			●					●		●	User interface

Table 2.2 Synthesis Matrix – summary of existing solutions

Since the majority of solutions are inclined towards bespoke application scenarios they are unsuccessful in demonstrating the addressing of low-level technical issues, e.g. hardware interfacing, computational load balancing and precise time synchronisation. Generally, solutions found in the literature related to multimodal DCIS were designed to address higher level aspects. An unstable link between the higher-level tools and low-level technicalities can always affect the performances of the higher-level tools built on top of the low-level fundamentals. Consequently, it becomes important to develop reusable, low-level tools that work closely, efficiently and transparently with regard to the low-level hardware, raw data transport and timing issues.

2.11. Summary

Contemporary DCISs use smart devices for capturing data and interacting with the environment. These systems are built for different purposes such as data capture, interactive interfaces, smart environments, etc. Building these systems with a variety of multimodal data and tools requires the addressing of many technical issues prior to achieving the intended and purposeful functionalities. The technical issues to be overcome are often similar amongst these diverse systems. Therefore, a need for a common basis of frameworks and systematic approaches that can be shared among these systems is clearly seen. Defining this as the problem statement and through domain knowledge drawn from literature, this thesis aims to establish a solution to address the technical necessities of building complex, multimodal DCIS. This chapter has predominantly addressed a thesis objective of reviewing the literature with the view to identifying the knowledge and technical gaps in the problem domain.

Chapter 3. Methodology

The literature survey provided in the previous chapter identified the key gaps and established the need for common set of rules, systematic procedures and reusable design to address the technical requirements of building a DCIS. This chapter outlines a potential solution and details the methodology used to develop the solution; the actual solution will be presented subsequently. Basically the methodology described in this chapter served as the primary methodology of this thesis.

3.1. Rationale for a software framework

The solution developed in this work is embodied in a software framework that provides a system-architecture and a collection of tools and techniques to address the requirements of DCISs. The domain of the framework was established and delineated to include the aspects of building complex, multimodal DCISs. The solution developed can be viewed from two perspectives, i.e. a piece of technical work providing core functionality and a software framework as a means for delivering the functionality.

The vision of the framework is to define the architectural structure and design parameters of a complex DCIS so that application programmers and system designers can concentrate on the specifics of their particular system. In contrast to an application developed to solve one particular problem, a framework is expected to be used across multiple application scenarios sharing a common problem. This emphasizes the reuse of design and knowledge in the framework across application domains. The systematic arrangements and structures defined in the framework produce norms which are not only reusable in multiple application scenarios but also standardise the process of building complex DCISs. This prevents the community of system developers and application developers from re-inventing techniques for building DCISs and producing ad-hoc solutions for every different application scenario.

3.2. Methodology of developing a framework

What are the characteristics required by a framework?

Theoretically, a framework is perceived as a re-usable piece of work or a semi-complete application that can be extended to build a custom application [9], [139], [140]. Further to design reusability, frameworks usually encapsulate useful framework-domain knowledge that is essential to build applications in an application-domain [141], [142]. A depiction of the systematic arrangement of components or an architectural skeleton to build applications is another important aspect usually embedded in frameworks. Characteristics required and demonstrated by frameworks have been analysed in various studies [9], [143], [144]. Gamma et al. suggested frameworks offer an overall structure to systems, partitioning the system into

modules, key responsibilities of components and the thread of control. Design parameters are predefined and encapsulated in the framework; therefore, system architects/implementers can focus on the specifics of their applications [143]. Fayad et al. defined modularity, reusability, extensible and inversion of control, i.e. framework code that has the thread of control and calls the application code when appropriate [145], as the primary benefits offered by frameworks to a developer [9].

How frameworks are developed?

Developing a framework is much more complex than developing an application since it has to be designed and optimised to successfully operate across various applications [143]. A framework typically evolves from the experience of finding answers to problems and developing solutions in a specific domain [146]. The generalisation of variable requirements across different applications is a common practice in the evolutions of frameworks [147].

The chosen development methodology

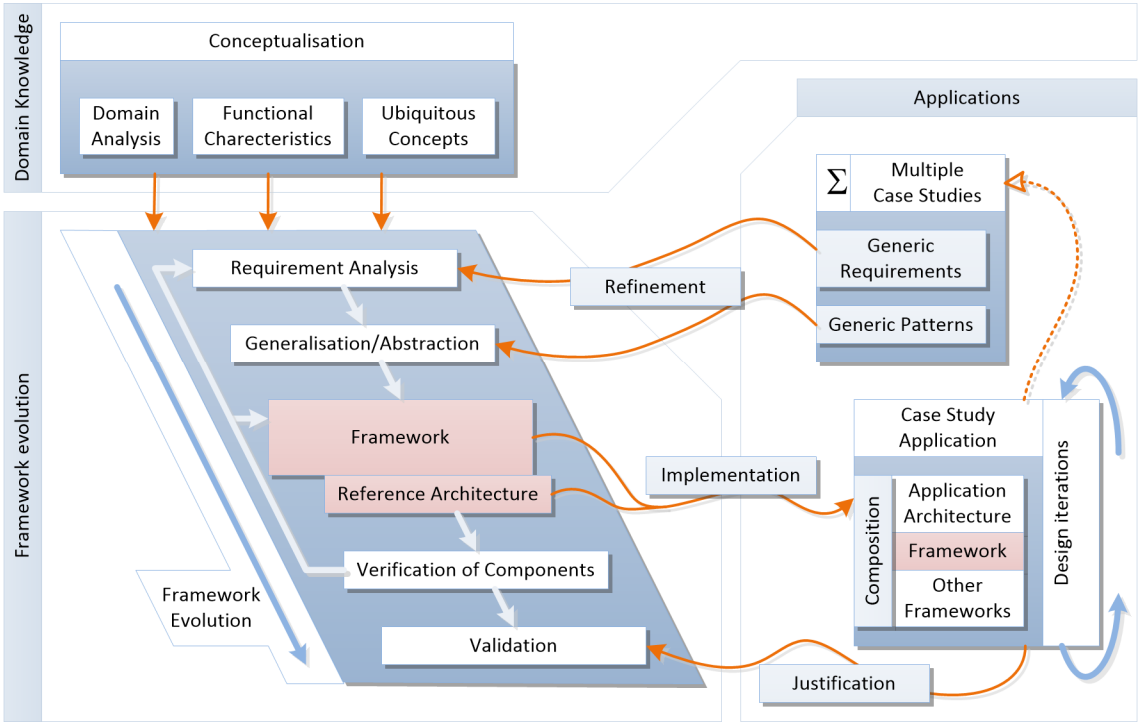


Fig.3.1 Framework evolution and the methodology

An insight into the standard characteristics of a framework design and how frameworks eventually evolve was derived from the literature of software frameworks and used throughout this work [144], [145], [147]–[149]. The principle methodology used to develop the framework in this thesis is tailored to the process of framework evolution based on domain analysis and experiences gathered by simultaneously developing applications. This methodology mostly aligns with the “General Framework Development Process” presented by

Mattsson [7] which is originally inspired from Wilson et al and Roberts et al [150], [151]. Fig.3.1 illustrates the methodology and the process of framework evolution, indicating various phases of the framework development cycle. The rest of this section elaborates on each phase and how they contributed to the overall framework evolution.

3.3. Domain analysis

Domain analysis was the initial development phase where the problem domain i.e. multimodal synchronous data capture and interaction, was comprehensively analysed through the literature survey. Existing software applications and toolkits were thoroughly examined for the features provided and how they handled the problem. Most of them addressed the problem partially and constrained the problem to a specific application scenario, in other words, in an ad-hoc fashion. The scopes of the existing tools were often constricted to specific applications, therefore multiple existing applications and prospective application scenarios were collectively considered in the analysis. This analysis provided an understanding of the problem domain, a general introduction to existing issues, how they are addressed and those that need to be addressed newly or differently.

3.4. Requirements derived from the domain analysis

Commonly encountered technical problems turn into the primary requirements of an effective solution. These requirements were derived from the literature and are briefly described below; basically they will be incorporated as a primary set of requirements for the solution being developed throughout this thesis.

1. **Integration of heterogeneous devices**

The solution should support integrating and interoperating devices from diverse functionalities, mixed manufacturers, dissimilar API/SDK. Assorted interfaces need to be unified so that uniform minimal access can be performed. Therefore different parts of the system can be developed independent of each other, still effortlessly interoperated.

2. **Multimodal data**

Similar to the diversity in devices, multiple data types needs to be handled homogenously regardless of modality. Data represented in both continuous streams and discrete events need to be incorporated. It is also necessary to support data transport in real-time from node to node, across various parts of the system.

3. **Temporal synchronisation**

The solution should provide methods to synchronise multiple data channels from asynchronous components of the system. The accuracy of the synchronisation may depend on the requirements of the application. Methods for synchronising data channels for real-time interaction and post-capture retrieval should be available.

4. **Ubiquitous and dynamic system**

A DCIS built using the developed solution may not need always to be fixed. Therefore, support for adding, removing and swapping components should be allowed in runtime, ideally in a plug-and-play manner. Devices in the systems need to be controlled programmatically and manually. Physical devices failing should not result the entire system failing fatally, the rest of the system should continue to function without interruption. Independent nodes and devices should be protected from unexpected mishandling. It is necessary to operate with commodity hardware, software, communication channels and general purpose OSs.

5. **Deployment configurations**

The solution should be flexible enough to be deployed in various configurations, for example, application processes in a single platform, physically separated distributed platforms and dedicated hardware platforms. It should support building both standalone systems and modular systems implanted into other operational architectures or infrastructures.

6. **Generic and abstract definitions**

The solution should be generic, enabling it to be ported in completely different arrangements with dissimilar devices and components. Custom, bespoke implantations will adopt and extend the abstract definitions. Maintaining the definitions in an abstract and generic fashion will enhance the reusability and extendibility across wider application scenarios.

3.5. Absorbing ubiquitous concepts

Introducing ubiquitous concepts into multimodal DCIS adds one of the key novelties of this work. The idea of ubiquitous and distributed systems was adopted and amalgamated into the framework. Consequently, a brief investigation of ubiquitous and distributed devices was carried out. Making no assumption regarding the data types and devices used in a DCIS makes the framework open to any new data types and devices those are unknown at the design time of the framework. Although ubiquitous concepts could add another level of complexity, embedding these concepts from the initial system design, offers a cleaner and stable design and needs less effort than introducing them in later stages of the design.

3.6. Requirements analysis

Both functional and non-functional requirements were identified from the domain analysis during the initial stage. The requirements mainly fall into two categories namely: DCIS related and system architecture related requirements. DCIS related requirements represent the core domain problem being addressed (e.g. multimodal device interfaces, temporal synchronisation); while the framework related requirements represent common framework characteristics (e.g. reusability, extendibility). DCIS related requirements were mainly derived from comparable and existing domain applications and any experience gained through developing custom, domain specific applications. Framework related requirements were initially taken from theoretical concepts and guidelines of framework evolution process [144] and later refined based on the framework implementation experiences.

3.7. Generalisation and abstraction

Requirements derived from similar domain tools and various domain specific applications are inherently dissimilar. Generalising the common aspects amongst this variability is a key step in defining the framework. Generalised requirements are defined in abstract terms, where they form framework hotspots. These hotspots can later be extended or customised during the implementation to meet particular application requirements.

3.8. The framework and reference architecture

The framework is composed of abstract definitions, recommendations and pieces of concrete implementations, i.e. classes and objects. In reality the actual, generalised/abstracted requirements and corresponding techniques and algorithms to satisfy the requirements are partitioned in to classes, structures and layers. These abstract features (or hotspots of the framework) will be extended or customised in the actual implementation to produce fully functional components. The framework inherently encompasses the key responsibilities of components and how they collaborate with each other.

It is practically impossible to design an effective and reusable framework simply based on the domain and requirement analysis. Although requirements and abstractions derived from an imaginary problem domain provide a decent starting point they tend to be inaccurate in the long run. Precise abstractions evolve from actually building applications using the framework and then re-evaluating its usability. Consequently, the proposed framework will be employed in a number of wide-ranging application scenarios for refinement, testing and validation.

A reference-architecture will naturally evolve based on the proposed and optimum usage patterns. This reference architecture can be used either to serve as the primary architecture in

applications or embedded inside the native architecture of the application wherever an effective and operational native architecture is available.

3.9. Case studies – Implementation examples

Concrete application examples will be used to determine which abstractions are efficiently reused and how the features of the framework adequately address the domain problem. Frameworks are often composed with other frameworks inside applications' architectures. The applications using the framework also undergo incremental development cycles in parallel with the framework evolution as shown in the Fig.3.1. Therefore requirements exhibiting a commonality among multiple applications will add up to the list of framework-domain requirements. Fig.3.2 illustrates the approach of generalising and abstracting the repeating patterns among multiple applications and then incorporated for framework refinement. Every refinement made to the framework thus will trigger an update to all applications. Modifications and code refactoring in the application caused by the framework refinements can be drastic or minor based on the severity of structural changes. However required refactoring in the applications will become trivial while the framework gradually evolves and matures.

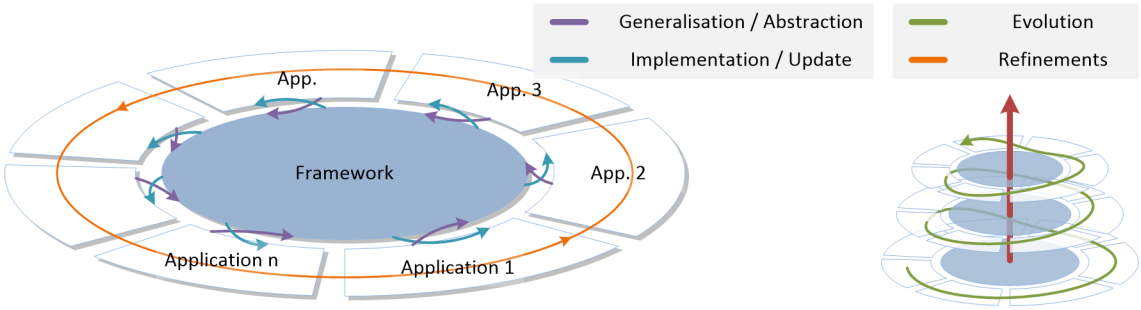


Fig.3.2 Repeating patterns among multiple application implementations are incorporated into the framework for refinement. Refinements made to the framework also trigger updates in the application implementations during the evolution.

Case studies from diverse scenarios will be chosen for the actual application implementations of the framework. Case studies will be selected from a variety of applications so as to reflect the requirements of diverse systems; therefore, they will possess a wide coverage in terms of applicability, functionality and inherent flexibility. In contrast to an application solving a specific problem, it is vital for a framework to cater for future requirements and corresponding refinements. The methodology chosen for this work accommodates plentiful of future refinements and it is assumed by default throughout the development. Impending requirements which are unknown at present will be appended to the list of requirements and will be addressed in successive development iterations.

3.10. Verification and testing of Components

The verification procedure will be focused into two aspects, namely: the ability to provide core functionality and an assessment of framework characteristics. The functionality verification will be performed in each cycle of the development and the outcome of the verification will be fed back into the requirements. The characteristic verifications will focus more on the structural design of the framework; based on this the framework will be improved so as to enhance reusability and reduce the effort of actual software implementation.

3.10.1. Verifying components and functionality

The framework will be evaluated and confirmed to satisfy the requirements drawn out from the initial domain analysis and case studies. Components and techniques will be justified to address the requirements by their intuitive design whenever appropriate. In other cases, quantitative metrics will be generated to verify that the design sufficiently meets the requirements. Special test rigs will be built to measure the performance and produce numerical data. Nevertheless, test cases will be embedded into the case studies, selected and effectively used whenever possible.

3.10.2. Framework metrics

Following to the verification on fundamental functionalities provided through case studies and specialised test rigs, summary metrics will be produced based on software code versions of abstract framework core and its extended solid implementations. These metrics are basically produced to establish complexity, reusability, extendibility, implementation and maintainability characteristics of the framework. This information can be used as a blueprint for future applications and therefore assist in choosing this framework for use.

3.11. Validation

The framework is validated as a complete system by its applicability to the scenarios pronounced in and across each case study. While applications in the case studies were used to fine tune the framework, they are actual examples of the framework being applied for problem solving in those application domains. Furthermore, employing the framework in any future applications not included in the case studies can be considered as demonstrating its applicability and providing further validation.

Beyond demonstrating the applicability and addressing the requirements in the application domain, the framework offers further benefits to the application domain. For instance, temporal synchronisation and dynamicity provided by the architecture delivers unprecedented ways to correlate the activities occurring in the environment with the captured information.

Consequently, the framework creates opportunities for new methodologies and techniques in each application domain, beyond building a fundamental DCIS. Examples of these new opportunities will be discussed with the associated case studies.

3.12. Presentation of the framework

The previously presented literature survey outlined a comprehensive critique of the problem domain (see: Chapter 2). The literature survey included a detailed domain analysis to find and critically evaluate the techniques already in existence and the potential aspects those are needed to be included in the framework. Subsequently, the problems identified from the literature are transformed into the requirements of the framework domain (see: Section 3.4). After the domain analysis the framework is presented in abstract terms (see: Chapter 4). In order to maintain its abstractness, it has been presented purely without any example illustrations. Although examples from the case studies inserted through the framework descriptions may help to improve elucidation, a pure presentation can serve as an untainted reference documentation of the framework. This approach is preferred and more suited in assisting potential users those who would be mostly focused towards their applications and not necessarily have about the case studies as examples.

Following the framework description, the case studies provide practical examples with and solid implementation details (see: Chapter 5, Chapter 6, Chapter 7, Chapter 8 and Chapter 9). They will describe how the framework is employed in each application scenario. The case studies are intended to demonstrate the validity of the proposed framework to solve the framework-domain problem of building complex multimodal DCISs within their own application domains. Each case study introduces its specific domain problem in addition to the framework related contents. This facilitates emphasising new opportunities delivered by the employing developed solution in each application scenario.

Successively, framework characteristic metrics are established based on the usage of the framework in case studies (see: Chapter 10). The novelties of this work are discussed and compared to the existing literature (see: Section 10.4). The experience of developing the framework and employing it in various practical scenarios established a better understanding about the future directions and required improvements; this information is provided towards the end (see: Chapter 11).

3.13. Summary

This chapter presented the methodology used in this work to develop a solution to address technical problems arise in building in complex, multimodal DCISs. The initial step in the

methodology was to perform the domain analysis; which has been provided in the previous chapter. The domain analysis covered various domain tools and diverse application scenarios. Diverse requirements derived from this analysis were analysed and the common aspects were generalised and abstracted. Along with the domain requirements, as a key novelty, the concept of ubiquity was introduced in the solution and amalgamated throughout the design specifications. The primary objective of the methodology here is to develop the solution in the form of an abstract software framework that enables building DCIS architectures. The following chapter will provide a detailed description of the framework design. Basically, the primary thesis objective of designing a framework is being addressed through these chapters. Multiple case studies will be used as actual implementation examples of the framework. These case studies help to refine the framework, verify components' functionalities and provide validation. Based on the case study implementations summary metrics will be produced, which is basically intended to serve as a blueprint to assist the potential users of the framework.

Chapter 4. Ubiquitous Integration and Temporal Synchronisation Framework

The previous chapters have established the domain knowledge, highlighted the challenges of building a DCIS using ubiquitous multimodal integrated tools and the methodology used to develop a framework. This chapter presents the Ubiquitous Integration and Temporal Synchronisation (UbiITS) framework as a generic solution to address technical issues in building a DCIS. Various components developed as a toolkit to systematically build a DCIS are detailed here. Techniques provided by the framework to handle the challenges of synchronisation are also explained in this chapter.

4.1. Fundamental use of UbiITS

The overwhelming reason for developing the UbiITS framework was to enable a simple yet efficient means to mobilise multimodal tools and quickly utilize the captured information in a DCIS. Accessing assorted devices directly is inherently complex, particularly when integrating multimodal devices. Time encoded synchronisation requires careful consideration and handling of the rudiments of packet (data) transactions. The UbiITS framework hides these complexities behind various layers of abstraction. Consequently users do not need to be concerned about details regarding the integration of the multimodal hardware and low-level interfacing issues but can concentrate on their immediate data processing needs.

The framework is presented as abstract layers and modular functional blocks, so that a DCIS can be designed using these modules as building blocks. These functional blocks are presented as a toolkit and can therefore be assembled in various systematic formations. A DCIS can consist of multimodal tools from data capture devices and fusion units to filters and interaction units, with each component operating on different platforms, APIs, algorithms and libraries. Therefore a complete DCIS can be constructed by extending and customising the abstract functional blocks and assembling them into a structure, specifically designed for an application scenario. Typically a software developer/architect (the designer of the DCIS) will assemble various APIs, libraries and code blocks as defined by the architecture to progressively launch the whole system.

4.2. Reference architecture, layers and components

Initially, a primary architecture is presented to identify the layers of a DCIS. This architecture makes the sections of a DCIS transparent and highlights the location of the framework in the system. This architecture is presented as a reference based on the recommended usage patterns of the framework and generally fits most DCIS configurations. However, the designer has the flexibility to derive a customised architecture for a particular application scenario if required. It also implies that the framework is suitable to be composed inside any external

architecture, for example a well-established native architecture for a specific application domain, given that it accommodates the basic concepts provided in the reference architecture.

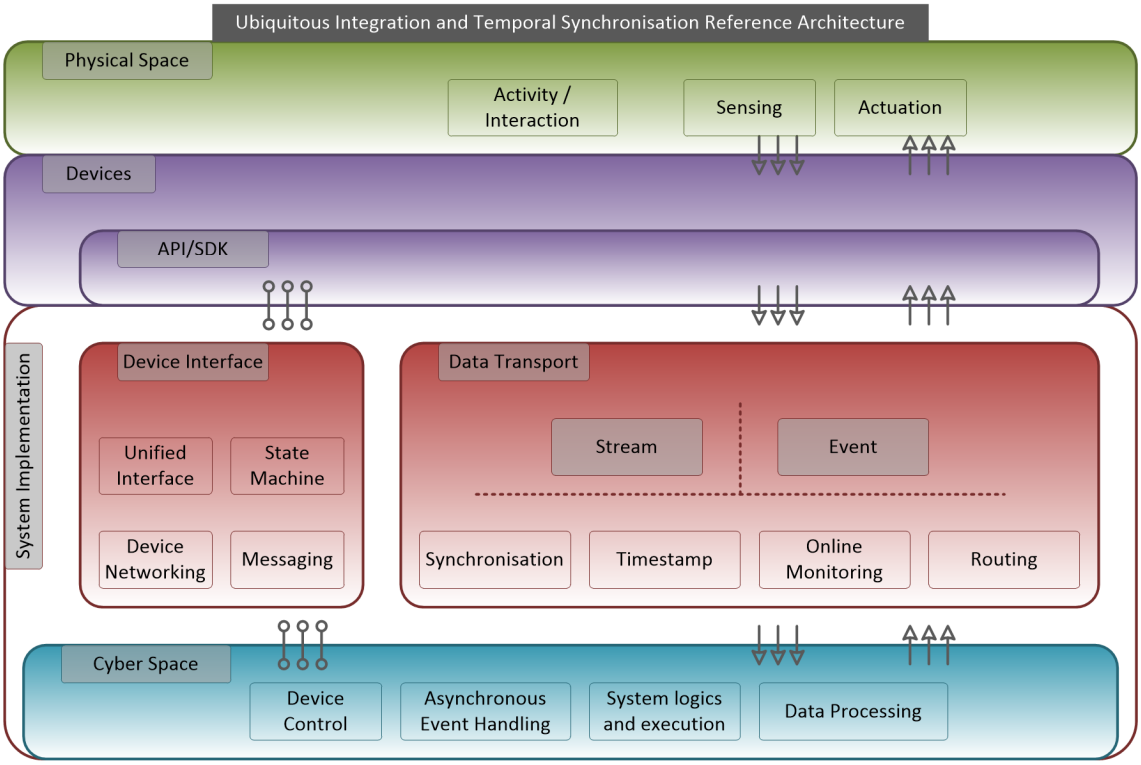


Fig.4.1 Ubiquitous Integration and Temporal Synchronisation(UbiITS) Framework and reference Architecture

Fig.4.1 provides a basic illustration of the reference architecture. The architecture extends from the physical activity to the computational and dynamic data/device handling. A description for each layer in the architecture is given below.

- **Physical space** – Where the actual, physical interactivity happens. Dynamic subjects, e.g. humans, living beings, and physical parameters, e.g. temperature, movement, etc., reside in this environment. The information is captured in this layer via sensors whereas interactions are performed with the subjects or environmental parameters are controlled via actuators. The actuation can be stimulus, feedback or a material with information content. The subjects usually perform an independent activity or react according to the input/feedback provided by the system, i.e. a closed loop.

- **Device** – A device is hardware or software unit that captures the information or interacts with the subject or environment. Usually a device is accompanied by an API or SDK. Software packaged with a user interface can be considered as a device since user activity can be captured and it also interacts with the user. Other kinds of devices

may not participate in capturing or performing a direct interaction in the system but contributes indirectly, for example storage, data compression, signal processing units.

- **System Implementation** – The system implementation is the functional organization in a DCIS. It predominantly serves the requirements of interfacing and integrating both devices and data with the execution/processing layer. Basically the system implementation links the physical space where the activity and interactions occur to the cyber space where the information is processed, computations are performed and decisions made in real-time. Various components used to design the system are defined in the framework and explained in detail later in this chapter.

- **Cyber Space** – Information captured using the devices is utilized by the DCIS in this layer. For instance, purposeful processing will be performed in this layer by a system that collects information from multiple sensors, fuses and processes the information, make decisions and provides feedback. Normally a developer designing the DCIS will decide how to process/handle the data and the ways to utilize devices and control/configure them to achieve the requirements. For a simple data capture, i.e. a storage system without any real-time interactions, there will be no work done in this layer; the data is simply routed to the storage modules provided by the framework.

4.3. Framework components - UbiITS toolkit

UbiITS framework is principally a set of generic functional blocks and techniques that primarily address the technical necessities of integrating various ubiquitous devices and transporting multimodal data. It further functions as a toolkit, a collection of fundamental modules designed to fulfil commonly required functionalities. Although devices are multimodal and corresponding data formats dissimilar, the modules are designed to be generic. The fundamental modules are typically abstract and needed to be extended and customised to each multimodal element. However there are number of components of the framework that can be identified as common amongst the multimodal elements and devices which remain readily usable regardless of the modality or device.

4.3.1. Programming concepts

The abstractness of the framework ensures it is generic in form and is maintained throughout the actual implementation by the use of object oriented programming concepts such as classes and inheritance. Two versions of the framework have been developed in C++ and C# languages. The higher level abstraction ensures that the framework independent from any platform or programming language. This allows adoption into any programming language

supporting object-oriented concepts, i.e. classes and structures. Programming peculiarities and styles often differ among programming languages, for example the object oriented concept of multiple inheritances is supported in some but not all language. Since the definition of the framework is generic and abstractly presented, the system designer can flexibly choose the implementation format while making use of the native features available for the particular application and corresponding platform.

Once a particular device or data type has been registered into the framework, i.e. the required modules of the toolkit are appropriately extended or customised, that device or data type can be reused in as many instances as necessary. This implies that device and data modality can be reused without any modification, either in multiple instances within the same system or in another system with a different configuration.

Device handling and data transport are important requirements in a DCIS and the framework provides distinct feature sets for each of these. Device interface components are provided to integrate and control assorted devices in the system and the data transport components are provided to handle the data routing between various parts of the system and related synchronisation. Constituents of each these feature sets are comprehensively described in the following sections.

4.3.2. Device abstraction and unified interface

A unified access interface is defined to provide the ‘ubiquitous nature’ of the framework such that the multiple capture devices’ control functions can be interfaced and accessed homogeneously. Having a unified access interface enables a device to be controlled independent of their underlying low-level functions, i.e. as a modular unit. This enables a device to be controlled by a procedure in another part of the system (i.e. execution/processing layer) which has no specific knowledge about the device. Generally, a wrapper interface is implemented to hide the device-specific function naming conventions and merges them if there are multiple function calls required to accomplish one specific function; for example loading API DLLs, initiating connections with the hardware and checking for device availability that can be merged into the ‘initialization’ procedure. Principally the unified access interface abstracts device specific function calls for data read (input) and write (output) into a universal interface. Primary functions and their conceptual definitions are listed in Table 4.1 Generally APIs/SDKs provide analogous functions to the unified interface; otherwise, the system designer has the flexibility to pick device functions and fit them into abstract functions, as long as they sensibly align with the conceptual definitions of unified interface.

Function	Definition - Description
Initialize	initiating communication with the device and configuration; thereafter it is ready to be used
Release	freeing the device and the API
Start	starting the actual data read/write
Stop	stopping the read/write
Reset	resetting the device in case of errors have occurred

Table 4.1 Unified accessed interface, abstract functions

Conversely, some features are unique for particular types of device only. For example, selecting the input channel, e.g. HDMI, DVI and Composite, is an exclusive feature for video frame grabber devices. The framework allows access to similar device-specific functions directly, bypassing the unified access interface. However, it is preferable to access device-specific features through a bespoke device-specific interface while accessing the generic functions through the unified interface. This enables multiple devices to be controlled via a centralized control panel without losing the rigorous control of the device.

Integrating every device into one single large program creates potential complications in a ubiquitous DCIS, especially when the system configuration changes, i.e. swapping devices with alternative devices or reconfiguring device dependencies of the system, etc. In this case a programmer would usually modify, duplicate or rewrite the code from the previous setup for the new configuration. The concept of a unified interface implemented in the framework allows devices to be used in a modular manner in a variety of configurations without rewriting or duplicating the code. An initial implementation of the unified interface for a device is always needed after which it can be reused to handle different or multiple data logging arrangements without any alterations.

4.3.3. State variables for devices

The unified interface also allows devices to run independent of each other. It enables devices to attach and detach at runtime, permitting the ubiquitous DCIS to change dynamically. This unique feature embedded within the framework is particularly useful since it means that the operation of the system will not be disturbed should a device fails or is removed, when a new device is accessible to join the system and where modifications in runtime are required. This is achieved by allocating a status parameter to every device which can be checked to determine the availability or status of a device. For instance, when a device abruptly becomes unavailable/inoperative, part of the system depending on the device can safely disable some functionality or alter the dependency to another device, instead of failing irrecoverably. This feature is ultimately intended to provide a seamless and invisible access to all components and resources in the system.

In addition to protecting the whole system from unpredictable behaviour the state machine also shields a device from falling into an unstable state by stopping it being mistakenly commanded (e.g. trying to use the device before it is configured appropriately, trying to access a failed device, etc.). Fig.4.2 illustrates the state machine including the unified interface functions triggering the state changes and primary state variables, which are listed in Table 4.2.

State Variable	Description
Dormant	device is not available
Not-Initialized	device is present but not ready for use
Ready	device is available for use
Running	device is capturing data
Dead	device is failed, unknown state

Table 4.2 Primary state variables for device management

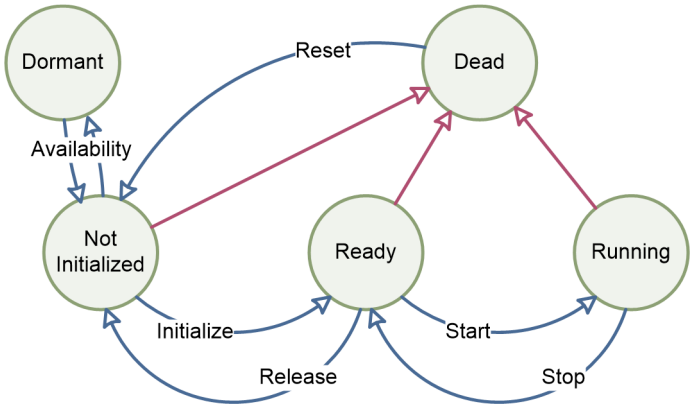


Fig.4.2 State machine with state changes triggered by the unified interface functions

The state machine defines the minimum states mandatory to support the rest of the system's operation. However, it can also be extended to include more states, i.e. sub-states. This enables support of more device specific states and therefore finer control over the device. It is necessary that each part of the system utilizing the extended states is aware of and pre-programmed to handle the extended functionalities or states.

4.3.4. Event and stream data types

The UbiITS framework classifies captured data into two distinct types, namely events and streams. An event is a discrete and spontaneous action occurring in the environment. Therefore the captured event will have an associated distinct timestamp. Streams are equivalent to a time series which comprise successive samples in time spaced at uniform time intervals. In contrast to events, samples in a stream do not necessarily have characteristic timestamps. Similar to the previously stated device interfaces, data handling modules in the

UbiITS framework are also generic and independent of the modality of the device. Therefore a designer can use the fundamental data handling components to hold any modality.

A fundamental event structure contains a unique ID and a timestamp. The unique ID is aligned with an incrementing counter and the counter value is what indexes the data to its corresponding event. This provides a separation between the data content from the event structure, therefore resulting in homogeneous event handling regardless of the actual modality of the event. This allows an exchange of abstract event information between parts of the system without transferring the whole data content of the event, leading to a compact and efficient transfer. In most situations, events are predefined and hence the event ID itself is sufficient to distinguish the events unambiguously. For instance a 32bit integer event ID can hold up to 2^{32} events, which is sufficient for most cases. However, if a designer decides to incorporate the additional data related to the event, the event structure can be extended to hold extra contents as well, e.g. descriptive text (string) of an event.

Unlike event data, stream data is typically diverse among modalities, are type specific and high in capacity. How the actual data is exchanged depends on the specific application, modality and designers' choices. UbiITS framework provides fundamental components to handle any type of stream. Sample count is a predominant feature used to handle data streams where it requires synchronisation between different types of streams, and between a stream and events. The sample count is a time related feature which represents the total number of samples collected at any one instance of time. This is continuously incremented when data stream samples are produced, one-by-one or in heaps depending on the method in device API for collecting samples. Sample count can be interrogated at any instance in time, usually for the purposes of synchronisation or monitoring a stream's characteristics. More details about how the sample count is used for synchronisation and stream monitoring are given in Section 4.6. FIFO buffers are also extensively used for stream data exchange and will be discussed in a separate section.

4.3.5. Data flow and real-time data access

Transferring data across various nodes of a DCIS is a fundamental requirement. Multimodal data transfer has to be transferred across components for reasons such as real-time data access, multimodal data fusion, storage, etc. Components in a ubiquitous system often work modularly thus independent of each other and hence asynchronously. The UbiITS framework implements data transfer using the model of nodes and links. This allows the data linkages to be effortlessly created or destroyed between compatible nodes. Data linkages can be set permanently, i.e. a hard coded system, or dynamically i.e. an active algorithm initiating

linkages based on demand and availability. Given that nodes are compatible, they can be linked together for data exchange. This enables nodes to operate independent of each other, such that compatible nodes can be interchanged or links can be redirected as required.

Fig.4.3 shows various configurations of data routed between nodes. Data originates from a source node and it flows towards a sink. Since components of a multimodal system run asynchronously it is crucial to assert that sources and sinks produce/consume data packets in independent clock pulses. Therefore malleability is compulsory in the data transfer between asynchronous nodes such that nodes can be freewheeled independent of each other. FIFO buffers are often used as a solution for the asynchronous data transfer [152], [153] and hence the UbiITS framework uses generic FIFO buffers, specially designed for multimodal data transport.

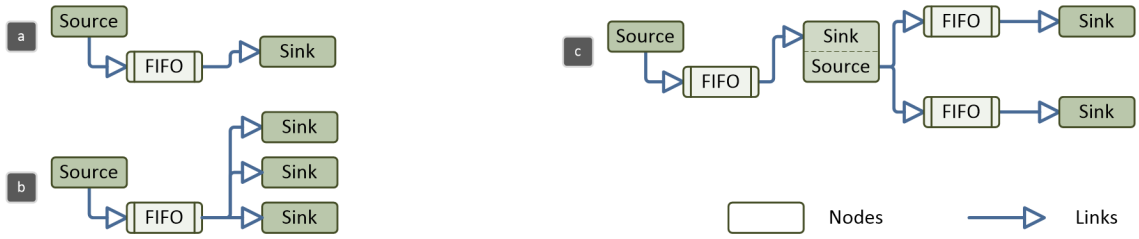


Fig.4.3 Example configurations of data routed accords components using the model of nodes and links. FIFO buffers play a key role in transferring data between asynchronous nodes. Data transfer across, one node to another (a), one to many nodes (b), chained nodes (c)

4.3.6. Asynchronous FIFO buffers for multimodal data transfer

The framework provides a generic definition for FIFO buffers for the purpose of maintaining the homogeneity among multimodal data types. Data packet formats are always custom designed for each application and vary widely among multimodal data types. Consequently, the storage/memory space for data packets are intentionally left out in the definition of the FIFO buffers. Instead, an index management structure is provided allowing the designer to define the data packet format whereas the FIFO index management structure is used to access data packets. This implementation delivers universal, yet lightweight and efficient means to handle assorted data types.

The FIFO buffers play a key role in data transport across asynchronous nodes. They are particularly used in the framework for stream and event data transport between asynchronous nodes. Furthermore they also play a vital part in solving synchronisation issues between multimodal channels such as introducing a controllable delay on streams as a technique for de-jitter. This is comprehensively discussed in Section 4.6.

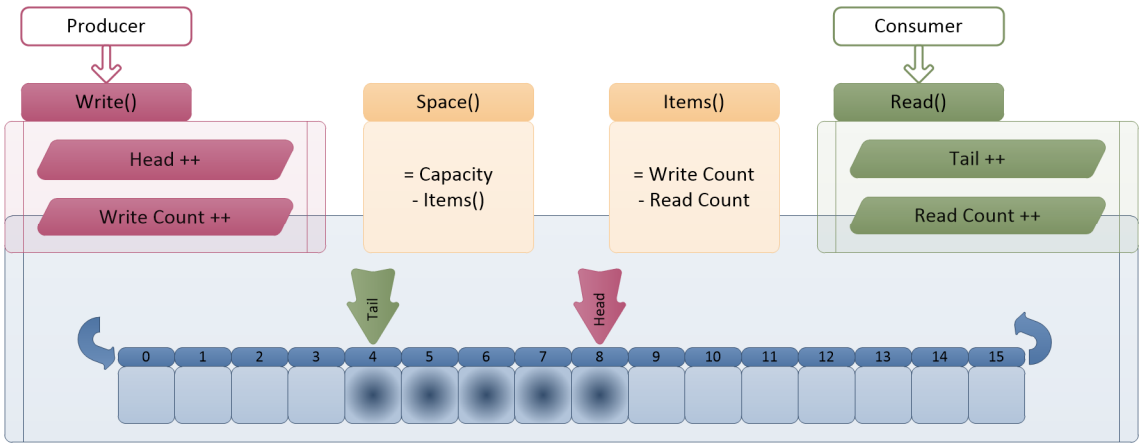


Fig.4.4 Single producer - single consumer, lock free and fixed size FIFO buffer implementation

The FIFO buffers are accessed by asynchronous nodes, i.e. by different threads using a shared memory in the process or separate processes during the operation. This results in the classic producer-consumer, thread synchronisation problem. As a solution the framework provides a single producer - single consumer (SPSC), lock free and fixed size FIFO buffer implementation [154]. Fig.4.4 shows the lock free FIFO implementation where the thread safety is guaranteed by preserving the mutual exclusion of variables being written; a producer and a consumer will never share the write access to a variable. The lock free implementation does not require any native locking mechanisms from the platform although it provides the highest performance. However, its nature and simplicity makes it vulnerable to creating deadlocks in the system when it is inappropriately programmed. The deadlocks can be eliminated by designing the algorithms to use non-blocking techniques; ensuring threads/processes using the FIFOs do not have their execution indefinitely suspended when a FIFO status is empty or full.

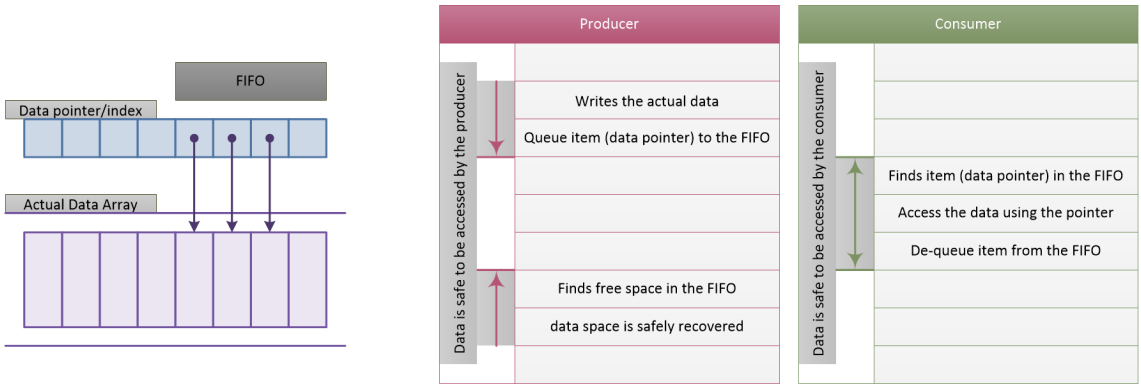


Fig.4.5 An example no-copy data exchange operation between a producer and a consumer

The FIFO index management structure enables no-copy operations between a producer and consumer. This is achieved by exchanging data pointers or indexes instead of the actual data (Fig.4.5). This technique is exceptionally efficient when high capacity data elements are exchanged. However, when the data elements are smaller, for instance if the actual data

element exchanged is equivalent to a native data type, e.g. native integer types of platform, float, etc., and exchanging the data pointers may not have any performance benefit.

UbiITS also caters for single producer–multi consumer (SPMC) implementation. This is useful when data generated from a node is to be routed across multiple nodes. The implementation is actually a derivative of the basic SPSC implementation. Multiple basic SPSC FIFOs are stacked to form the SPMC implementation, where each consumer will have a SPSC FIFO to read from but the producer calls a modified function to write to the SPMC FIFO. This modified routine differs from the basic SPSC implementation in that it checks all sub SPSC FIFOs' free space and takes the minimum value. In this manner the producer does not overwrite the elements in the FIFO until all consumers have consumed it. The previously described no-copy operations remain valid given that all the consumers have read-only access to the data and finish using the data before the consumer reclaims the data storage space.

In addition to the primary features described, the FIFO buffers are equipped with additional features to support the synchronisation between data streams. They are designed to provide information about buffer over-run and buffer under-run; therefore the runtime behaviour of connected nodes can be examined. For example, a node producing samples at a rate higher than the rate consumed by the paired node will lead to a buffer over-run. Similarly when the samples are produced by producer node in a lower rate, it will not keep up with the consumers and lead to a buffer under-run. These features are useful when the system behaviour is needed to be monitored online. A further discussion about online monitoring is given in the section 4.6.6.

4.4. Parallel execution and deployment configurations

Previous sections described about making use of the abstract components provided by the UbiITS framework to create customised device and data modules. Although the reference architecture and the framework outline an architectural skeleton for the system these modules must be compiled and deployed as an executable application in order to be a functional system. Various configurations of functional systems and how the independent modules physically execute are now described. The first deployment configuration relates to multiple devices and data handling nodes within an application. Thereafter, a combination of multiple detached applications with their own set of devices running on isolated processes/platforms is described.

4.4.1. Standalone deployment

A standalone application integrating multiple devices and data handling nodes is a simple and basic deployment configuration. The predominant characteristic of this configuration is that all modules live in the same process and hence share the same virtual memory address space. Serving modules in a timely manner is a principal task in a DCIS; for example polling data from devices, removing samples from buffers where they are being delivered, processing data in real-time, storing data to disks, etc. Each of these tasks incurs different time priorities and their computational loads also differ. Therefore it is crucial to serve the needs of each module in real-time priorities. Running concurrent threads within a process is the standard practice to satisfy parallel multi-tasking scenarios. In platforms without any operating systems, e.g. embedded systems this is handled through interrupt service routines. However, the real-time scheduling of the tasks is highly dependent on the underlying platform. Therefore the nature of scheduling and controllability differs across the underlying operating systems and platform, e.g. number of hardware processing cores, availability of RTOS and scheduling disciplines. Generally designers make the final decisions regarding system behaviour and performance based on their judgement of the computational loads involved and required priorities.

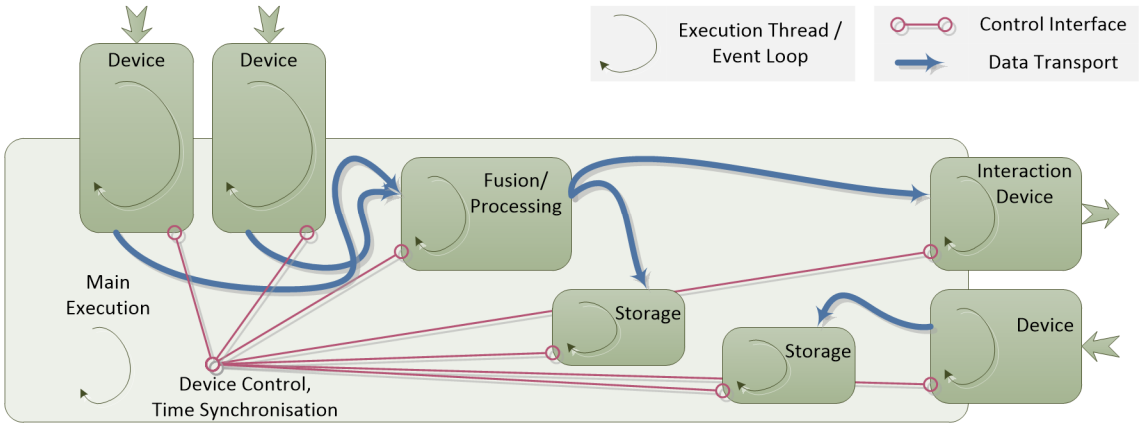


Fig.4.6 Standalone deployment of device and data modules. Modules are served in parallel with multiple concurrent threads

The structure and components of the UbiITS framework inherently assert parallel execution and multi-threaded deployment so that the modules can be served in a well-timed fashion (Fig.4.6). Typically each module is served in a dedicated thread; however considering the limitation in the maximum number of possible parallel executions it is mandatory for every module to yield its execution so that other tasks get time to execute. The scheduling of parallel tasks can be flexibly chosen by the designer based on the availability and suitability. For example one can use multiple timers to serve modules and run event loops per thread, e.g. Qt, timers and event loops. Using event loops and timer might provide a hybrid mode of time-

sharing and multi-tasking threads, hence overall number of required threads is reduced. In an RTOS a prioritised pre-emptive scheduling policy can be enforced so that each module gets time to do the service based on the priority. This is analogous to timer interrupt handlers with priorities in embedded systems.

4.4.2. Asynchronous function calls and messaging service

The FIFO modules described in section 4.3.6 are designed to work with parallel access and they primarily resolve the problem of data transport between two nodes (i.e. data modules) that run asynchronously. Yet, in most cases it is usually unsafe to access the device SDK and underlying device in multiple threads which might otherwise result shared memory access violations and inconsistent states in the SDK or underlying device. Therefore, it is important to call the device functions in a single thread where this thread virtually 'owns' the device and mediates all requests to/from the device. It is particularly more complicated in embedded systems where the actual hardware is accessed or maintained directly using the module. Several operations such as sequential read/write and accessing busses should not be interrupted and therefore careful considerations about thread scheduling and timing are needed. Thread synchronisation, locking mechanisms and atomic operations may be needed in some cases, depending on the specific case.

The dynamic control of devices requires the calling of their unified interface functions from various parts of the system, where the callers often run in asynchronous threads. However, the unified interface functions should not be called directly since it is only safe to call device control functions from a single thread as mentioned earlier. To address this issue the UbiITS framework offers a simple messaging service across devices. Messages can therefore be processed and the unified interface functions called only by the thread having ownership of the device (Fig.4.7).

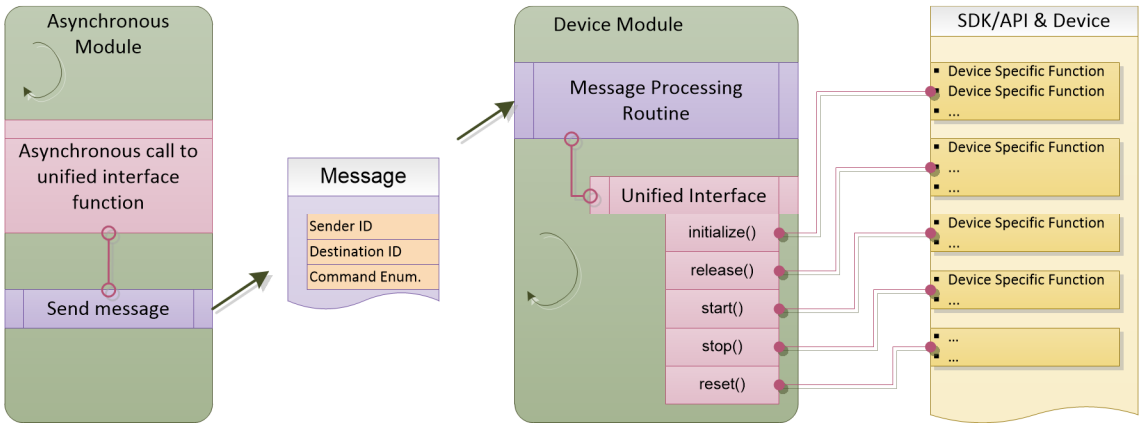


Fig.4.7 Messaging service for asynchronous calls of device functions

The message format is kept simple and includes sender ID number, destination device ID number and an enumerated value for the command being sent. For instance, if these parameters are of 4 byte sized, then they support 2^{32} values, adequate to cover for device/module IDs and command enumerated values in a typical DCIS. If the functionality needs to be extended other than calling the unified interface functions, the designer can extend the enumerated values.

The exchange of messages across asynchronous modules is usually performed via FIFO queues such that the sender puts the message in the queue and receiver picks it up from the buffer. However, based on the development toolset used to build the system the asynchronous messaging service might be substituted by the toolset's features which support asynchronous calls. For example, the signals and slots mechanism supports sending events across objects living in separate threads in the Qt Framework. Similarly delegate and invoke methods in the .NET framework support asynchronous calls. In both cases, whether using the message queue in the UbiITS framework or the asynchronous call-back features provided by the development toolset the fundamental concept remains similar to a queuing up mechanism. The latency on the asynchronous function call is dependent on how often the message processing routine checks for new messages in the queue.

4.4.3. Distributed deployment

Distributed deployment is more common in building a DCIS where multiple platforms are involved. Contrary to standalone deployment, the modules will not share the same memory address space. Multiple independently running processes, possibly in separate platforms, form the whole structure of the system. However, to function as a whole system these independent processes need to be interconnected using ordinary network connections and standard protocols such as Ethernet and TCP. This configuration can also be inferred as an interconnected network of multiple standalone deployments.

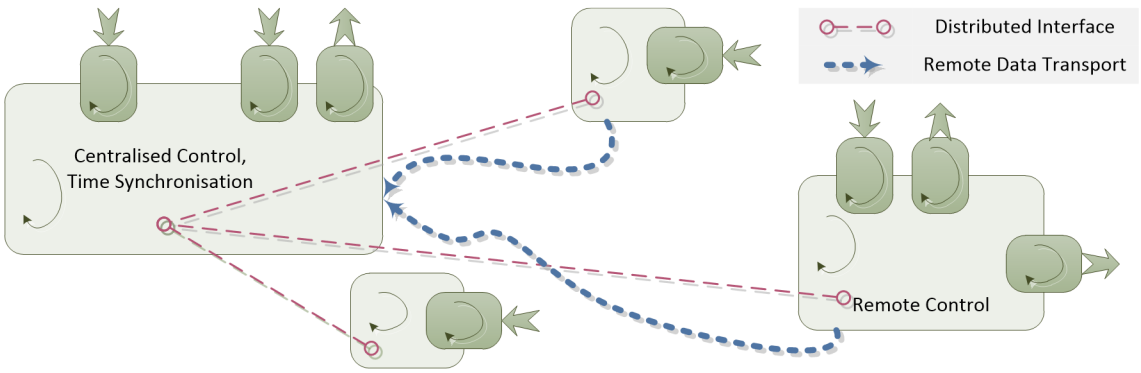


Fig.4.8 Distributed deployment of multiple interconnected standalone applications

Remote control interface and data transport are the two connectivity issues that require to be addressed in this configuration (Fig.4.8). A control interface is normally needed when devices need to be dynamically controlled from a remote location. The messaging service introduced previously and used within the standalone deployment can be effectively adopted for such configurations. The assertion made in the framework that all modules execute in parallel makes it possible to use the same techniques both in standalone and distributed deployments.

For instance if TCP connectivity is established across distributed deployments, unified interface functions of a remote device can be called by means of tunnelling the messages via the TCP connection. The same connection can also be shared for data transport and synchronisation purposes; however, a packet format needs to be defined for transmission such that it supports holding all required types of transmission packets. Bandwidth and latency issues play a significant role in transmitting packets across remote sites and therefore careful consideration must be taken when transmitting multimodal data. The diversity of networking techniques (e.g. TCP, Bluetooth, Serial port) and the characteristics of multimodal data (e.g. size, constant frequency, spontaneous events, allowed latency) mean that it is not possible to define an efficient and universal transmission packet format. Consequently, the system architect shall choose an appropriate packet format.

The distributed configuration can be in various arrangements based on the specific DCIS. For example, every isolated process can be connected to one centralised process or various processes can be connected to each other, e.g. star, mesh, bus, etc. topologies. Wrapping up a remote device or a data node into a virtual host module is a possible way of connecting remote devices. Host modules reside in the local process, so that they can be accessed directly and eventually the remote devices accessed through them.

4.5. Event driven system – Asynchronous and distributed events

Combining the two distinctive features of the framework, namely the data handling nodes and asynchronous function calls provides the opportunity to construct an event driven DCIS. Fig.4.9 shows how physical activities are converted into system events. This is a substantial feature indicating that the system can operate as an event driven system, not only responding to the programmatically generated events but also to the physical activities. When physical activities are captured as events they can be converted into system events and thereafter handled by an asynchronous event handler. A physical activity captured as stream data can be converted into event data when the data stream is processed for event detection, e.g. when a threshold is exceeded or a pattern is matched.

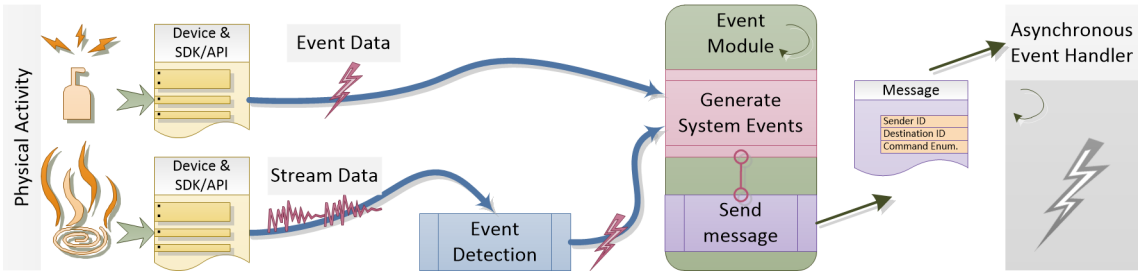


Fig.4.9 Event Driven System: physical activities transformed into system events

Generated events often have to be handled asynchronously, i.e. separate threads, so that the data handling nodes can continue to work uninterrupted. Events are required to be registered and routed by an event router where more than one part of the system is listening to systems events. Typically, an event handling mechanism provided by the development toolset (e.g. Qt signals and slots, .NET Events) will be in operation and therefore can be used effectively in place of a new event routing mechanism.

Event handlers may be used to produce chained events in response to an original event then the response event can be carried to an actuation device to perform a feedback activity. Alternatively, the event handlers may try to modify the system configuration, such as starting/stopping a device. This enables the DCIS to adapt its characteristics in real-time according to the captured data. Registering and routing system events in a distributed environment is complex since various connectivity solutions and development toolsets might be involved. However, distributed events are advantageous since they can be fired from remote locations and the entire system responding to those events by means of physical actuations. An event driven mechanism with asynchronous event handling and distributed events enables the building of a DCIS which is ultimately dynamic and responds to the physical activities.

4.6. Synchronisation techniques

The UbiITS framework is equipped with number of primary synchronisation primitives to achieve the synchronisation of multimodal data channels in real-time and for post processing. The functionality provided by the framework components for synchronisation is categorised as follows.

1. Time stamping and chronological ordering of events.

2. Synchronisation of multiple data streams.

Components and corresponding techniques used to achieve time critical synchronisation are detailed below.

4.6.1. Timestamps

The framework defines an abstract clock module to provide timestamps. The clock module is usually a singleton object for a process running on a specific platform. Therefore all modules use the time provided by the clock to timestamp the event or sample. Usually in PC based platforms, there is at least one high precision hardware clock provided and a clock module is essentially a wrapper for this native clock. The system implementer will generally provide a solid implementation for the abstract clock module provided in the framework, making use of the platform specific clock. In circumstances where there is no clocking mechanism available (e.g. embedded hardware with no dedicated hardware clock) a simplistic revolving counter pseudo clock implementation is provided in the framework. This pseudo clock implementation needs to be ticked appropriately by some mechanism, e.g. interrupt service routine (ISR); hence the ticking accuracy can affect the time provided by the pseudo clock.

The standard timestamp defaults to 64 bit integer across the framework which is basically represented in the UNIX time format. Accessing the hardware clock in synchronous threads is not problematic since it is a read-only operation. In platforms where it is not safe to interrupt the clock access, the time read must be protected by thread synchronisation mechanisms.

As stated previously in the section 4.3.4 time stamping is an essential element for marking the temporal location of events occurring in a DCIS. These events are captured from multiple sources and given that they all share a same time origin, i.e. a centralised clock, they can be aligned temporally and chronologically. Similar to events stream type data can also be time stamped, thus associating a timestamp with each collected sample. In this case a timestamp does not need to be treated as a separate entity; data structure designed hold the sample data can be extended to hold the timestamp data if required. For this reason UbiITS framework does not integrate timestamps to the stream typed data.

4.6.2. Clock synchronisation

Modules in the same process/platform share the same clock source; therefore, the timestamp synchronisation problem does not exist. However, in a distributed deployment configuration, timestamps might be acquired from various clocks. In this case it becomes necessary to ensure the temporal legitimacy of the timestamps. The framework provides several clock synchronisation mechanisms derived from Network Time Protocol [91] and Reference Broadcast Synchronisation Algorithms [155]. The clock synchronisation methods are abstract definitions and therefore suitable for any connectivity media, e.g. TCP, UDP or Serial. Once the offset of distributed clocks are estimated they can be synchronised beforehand or alternatively the offsets can be used to adjust the timestamps in another distributed platform. Fig.4.10

illustrates how events time-stamped by two distributed clocks should be treated in another platform with associated offsets.

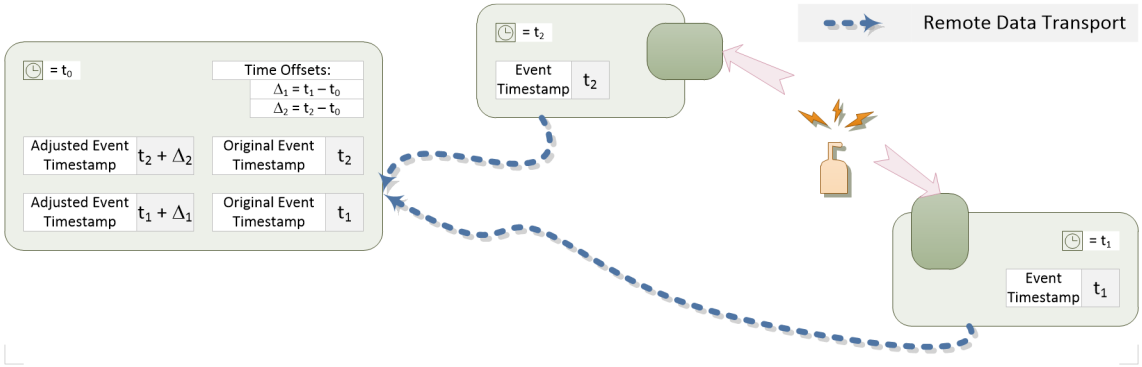


Fig.4.10 Time offsets in a distributed deployment configuration and compensated timestamps across distributed nodes

4.6.3. Latencies in data channels

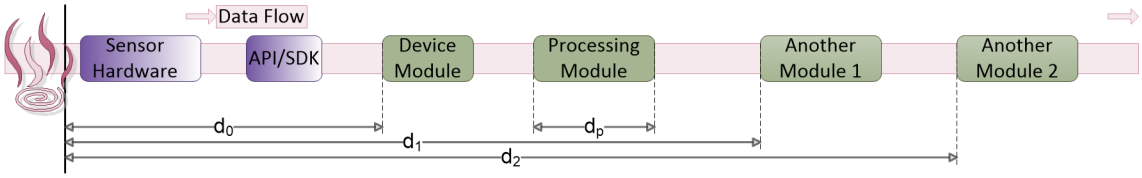


Fig.4.11 Data flow through various components in a data channel and associated latencies are indicated

Section 4.3.4 defined that physical activities can be captured as an event or a stream by the framework. There is always a temporal latency between the actual physical activity and the captured data associated to the physical activity (Fig.4.11). This latency can be considered as the time difference between the instantaneous physical activity and its corresponding data (an event or data samples in a stream) that is being processed for use. In reality it is impossible to completely remove this latency though it is possible to reduce it by having faster sensing and transport. The framework provides parameters to characterise this latency; therefore, instead of disregarding the latency it can be measured, catered for and compensated.

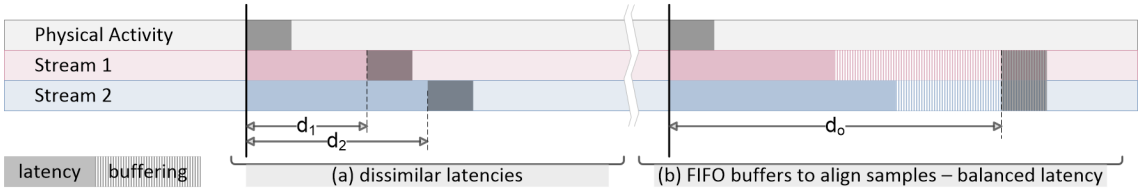


Fig.4.12 Disparate latencies involved in samples routed through various data paths. Buffers are introduced in the data streams to align samples in real-time

Furthermore, the samples collected via miscellaneous devices are routed through various data paths thus possessing disparate latencies (Fig.4.12, a). It is essential to temporally align

multiple data streams so that correlations made between data streams and the actual physical activity is valid. The framework approaches this problem with two following views:

1. Synchronisation for real-time data access and fusion.

2. Synchronisation of captured data in post process.

FIFO buffers are used in the paths of data streams to introduce extra delays in real-time to balance dissimilar latencies such that samples from different streams can be temporally aligned (Fig.4.12, b). The number of items (samples) available in a FIFO buffer can be used to trigger the data consumption from the buffer, i.e. controlling the latency. It should be noted that introducing extra buffers increase the overall latency, however this inevitable when a precise online synchronisation is required. Therefore the extra buffering should be kept minimal to reduce the effective latency.

Similar to stream type data, events type data also possesses latency issues. However events are normally aperiodic and therefore the system cannot rely on buffering number of samples to synchronise them with other streams or events. Latencies in events are directly related to the time offset, whereas in streams it can be related to time via the number of samples and their frequency. Latency/offset compensation parameters are incorporated for every data channel in the UbiITS framework. These parameters can be used up by any module which is sensitive to synchronisation across data channels (e.g. online-fusion units) to compensate for temporal offsets. However, it is not necessary to use these parameters when the synchronisation is not critical, (e.g. data storage units). When interrogating the offset parameters in a distributed deployment additional consideration needs to be taken since the distributed data transport also causes extra delays. Fig.4.13 shows how additional latencies need to be counted in a distributed deployment.

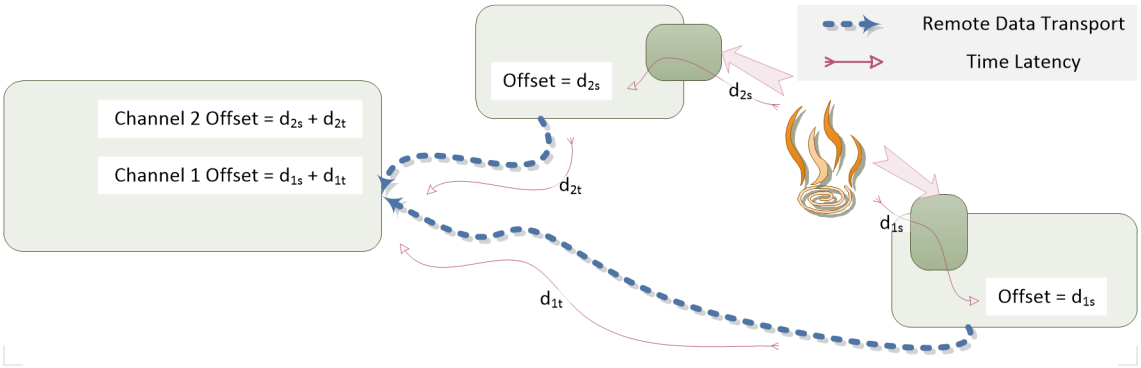


Fig.4.13 Latencies due to distributed data transport and dissimilar offsets in distributed deployments

4.6.4. Automatic latency measurement

Ideally latencies in device specific data channels should be available explicitly from the device/API specifications or could be evaluated based on specifics provided by the device/API manufacturer. Latencies due to internal components, e.g. processing, buffering, etc. can be evaluated by scrutinizing the algorithms and respective codes. However, based on the inaccessibility or difficulty in accessing this information in most circumstances, the UbiITS framework proposes techniques to independently estimate them.

A simpler way to measure latencies across data nodes is to transmit synthetic data across the source and destination nodes in known times. It is crucial to have synchronised clocks in both source and destination nodes, therefore the sending and receiving timestamps can be matched. This procedure functions efficiently in measuring latencies inside the framework's application realm; within this realm all the modules have access to some kind of a clock (Fig.4.14, 1). However this procedure turns out to be useless in finding the latency between the actual physical activities and when the corresponding signals initially become available in a framework module (Fig.4.14, 2). The main reason for this is that internal clock references within the system cannot be used to measure the precise instantaneous time of an external physical activity.

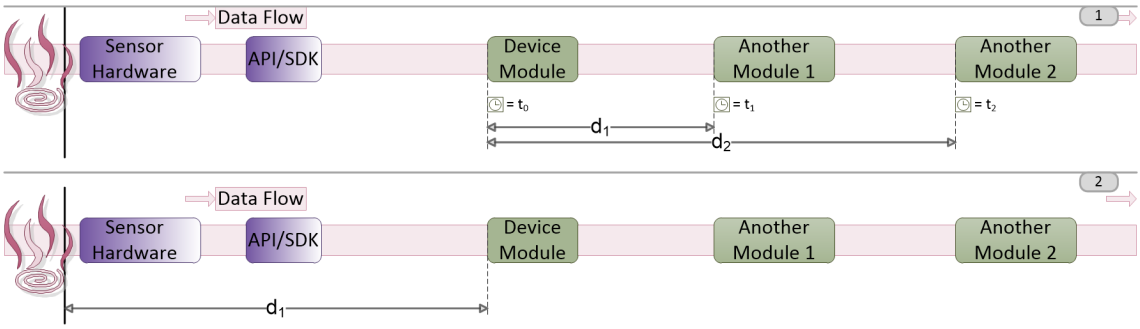


Fig.4.14 Measuring latencies within framework modules (1) and the latency between actual physical activity and the corresponding signals becomes firstly available in a framework module (2)

Measuring the latency of an external activity with reference to an internal clock is a convoluted procedure. To solve this issue a sophisticated technique is proposed in the UbiITS framework which can be used to measure the signal latencies of external activities. This method uses a dedicated embedded hardware as a calibration device to induce unambiguously identifiable data signatures into the data channels when commanded by the system. These induced synthetic signature patterns mimic a real external activity, and travel through the same signal path. On the other side the device module which is integrated in the system will receive and detect the synthetic pattern.

The step-by-step activities involved in procedure is given below and illustrated in Fig.4.15.

1. The system sends the command to the calibration device to induce a data signature in the signal stream.

2. The calibration device induces the synthetic patterns (i.e. data signature) in the signal stream and simultaneously (within a negligible time) acknowledges the command.

3. The induced signal is received and detected by the system.

Finding a suitable method for spawning the data signature into the data stream and detecting the signature are key requirements in this technique. The method for inducing a signature varies depending on the type of data modality. Sensor properties and the availability of actuation methods for inducing synthetic patterns also need to be considered when selecting a method. Spawned signature patterns require to be explicit to create 'spikey' pulses in the data channel so that the relevant algorithms can recognise the changes and detect the pattern unambiguously with the sufficient temporal exactness.

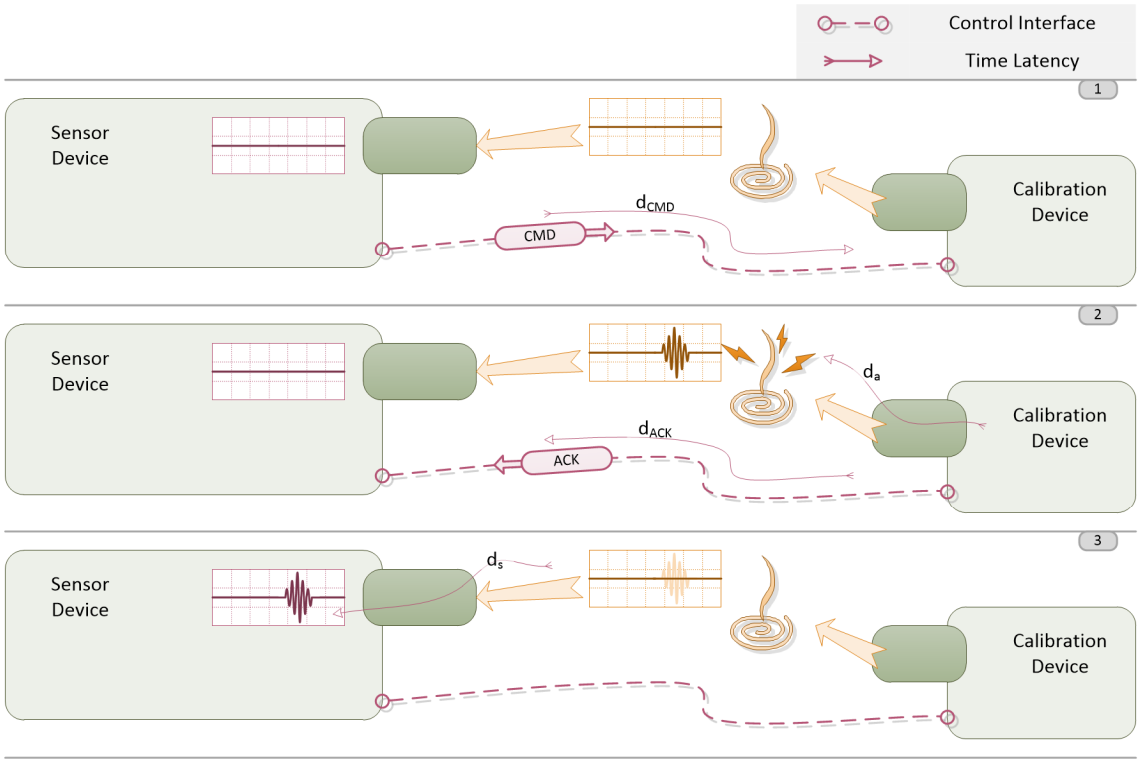


Fig.4.15 Measuring the latency of an external physical activity using a dedicated embedded, calibration device

The combined latency in signal paths can then be measured by obtaining the internal system timestamps for the following activities. The activities and corresponding timestamps are given below.

t_0 - command sent

t_1 - acknowledgment received

t_2 - induced pattern detected

The time diagram in Fig.4.16 illustrates the activities, time stamps and latencies involved.

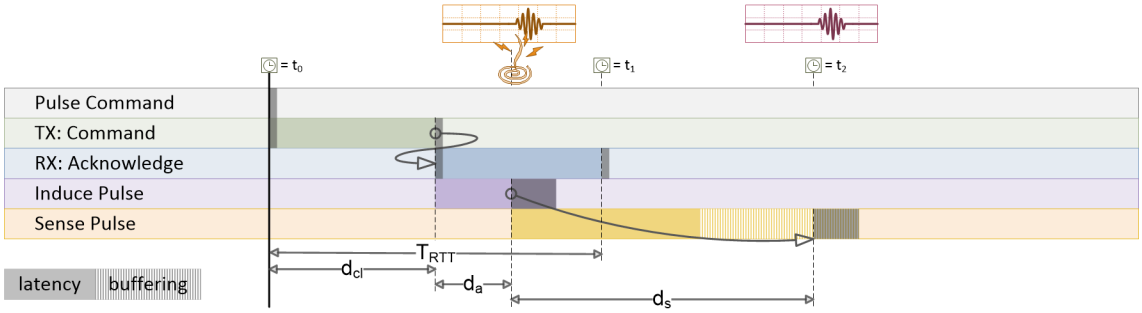


Fig.4.16 Time diagram for Measuring the latency of an external physical activity using a dedicated embedded, calibration device

Generally the round trip time of the command-acknowledgement cycle (d_{cl}) can be assumed to be symmetric. Therefore the command latency can estimated from the round trip time (T_{RTT}) as follows.

$$d_{cl} = \frac{T_{RTT}}{2}$$

$$T_{RTT} = t_1 - t_0$$

$$d_{cl} = \frac{t_1 - t_0}{2}$$

The signal latency can be estimated by the actuation latency in the calibration device (time taken to induce the signature pattern in the data channel), the time of induced signature being detected by the system and the command latency.

$$d_{cl} + d_a + d_s = t_2 - t_0$$

$$d_s = \frac{2t_2 + t_1 - t_0}{2} - d_a$$

In most cases the actuation latency is in negligible orders compared to the signal latencies involved. However, it can be quantified based on the embedded hardware design and physical properties of actuation, if required. Multiple signal latency measurements can be taken by firing calibration cycles in a series over a period of time. A better estimate of the latency in a data channel can be calculated by averaging the multiple signal latency measurements.

The dedicated embedded platform is used here for providing deterministic timing so that all execution timing inside the calibration devices are known and controllable. Inaccurate and non-deterministic timing issues are common in general purpose platforms and general purpose operating systems. An embedded hardware with low latency platform is a preferred solution for this. For example, an embedded programmable device driving communication and signalling peripherals directly without an operating system is preferable. Accessing hardware registries and interrupt service routines generally provide low execution times and predictable behaviours in task scheduling.

Typically latency/offset parameters can be fixed to a constant value based on the results from a test rig implementation. However if advanced or much tighter synchronisation is required, advanced algorithms (e.g. moving/weighted averaging, predications) based on the specific application can be employed to modify these parameters dynamically.

4.6.5. Overall interaction latency

As well as the presence of the latency in data paths, latency between the physical activity performed and the interaction is inevitable. Basically a time delay is always present when a change in parameter occurs and the system reacts to that change. Increased real-time processing time results in increased interaction latency. Although FIFO buffering helps in balancing the latencies across channels, it nonetheless increases overall latency of a reactive interaction. Fig.4.17 shows the latencies involved when a system reacts to the captured data.

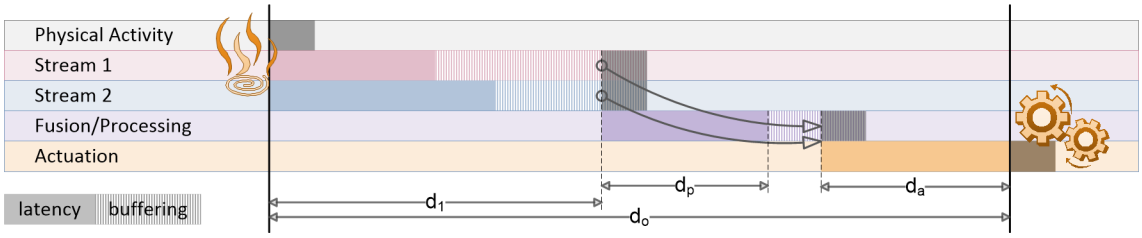


Fig.4.17 Overall interaction latency, includes signal latencies, buffering and processing delays

4.6.6. Runtime monitoring of sampling

Up to this point, how the UbiITS framework considered the latencies in data paths has been discussed. For any DCIS deployment it is important to understand that latencies often change dynamically, e.g. due to network dynamics and task scheduling inaccuracies. Because of this, node-to-node sample delivery frequency may also suffer jitter. Hence FIFO buffers perform a dual function here; helping to control/balance the latency and providing a cushion effect against the jitter Fig.4.18.

Fig.4.18 FIFO buffers used for providing a cushion effect against the jitter

Monitoring the buffer levels periodically can reveal the dynamics of the synchronisation and help to measure synchronisation related parameters such as jitter, buffer over-flow, buffer under-run, etc. In a real life DCIS fatal scenarios such as device failure or a loss of connectivity can occur and affect the synchronisation of data channels. Extreme scenarios may cause a DCIS to go beyond its limits and produce erroneous results. Detecting the extreme and out-of-control synchronisations in real-time can help designs to take intelligent, safety measures such as shutting down the functionality or nullifying the generated output. Monitoring synchronisation related parameters enables identifying synchronisation problems in runtime, recording them for later use and resuming the normal operation of the system once the parameters return to an acceptable range. Logging the sample delivery rate, jitter, etc. provides the ability to evaluate the runtime capabilities and manage the resource availability to handle a specific task.

Runtime parameters related to synchronisation can be captured periodically like capturing normal data samples. The meta-samples associated to the synchronisation are treated identically to any other samples in the system. The data handling modules in the framework allows these meta-samples to be represented in a data stream, and therefore they can be transported across modules, used and stored as necessary.

4.6.7. Automatic forced syncing intervals

Instead of assuming a constant delivery rate of samples in a stream, which is affected by the runtime characteristics, e.g. device failure, jitter, etc. a better method for synchronising stream samples is proposed here. Polling the sample count of every data stream in a periodic manner can force all the streams to be synchronised at periodic intervals. In an ideal case the sample count in every data steam would increase linearly; however this is affected by the variation in sample delivery rate. Sample counts polled at periodic intervals can reveal instantaneous frequency. Calculating the instantaneous frequency of a stream from consecutive sample counts and timestamps at polling is given below.

$$fi_n = \frac{SC_n - SC_{n-1}}{t_n - t_{n-1}}$$

Parameters associated with a poll - n:

SC_n - Sample count

t_n- Time

fi_n- Instantaneous frequency

A periodic timer can be used to raise synchronisation events automatically, i.e. trigger the polling of sample counts and estimating the instantaneous frequency. Shorter intervals will generate more accurate synchronisation points but increase the communication overheads and the size of the synchronisation data. The final synchronisation can be achieved by timely aligning and scaling data strips between synchronisation events. The data between synchronisation event points can be interpolated with the instantaneous frequencies.

4.6.8. Post capture synchronisation

Synchronisation requirements for post capture are slightly different to real-time. Samples are generally stored in to a storage device and thereafter during post processing streams realigned as required. Therefore buffering samples in real-time to temporally align data streams is not necessary in this case.

Compared to the naive approach of synchronising sample streams based on the beginning and end of the recording, the UbiITS framework was designed to handle complex DCIS comprising many different hardware and software components. Therefore it must be prepared for frequency variations and dynamic synchronisation characteristic changes in the runtime. Monitoring the synchronisation characteristics in real-time as stated in section 4.6.6 and 4.6.7 are essentially designed to improve synchronisation accuracies. Accuracies can be improved by dividing the data streams into periodic time strips and then matching the samples from temporally corresponding strips of multiple data streams. Basically shorter time strips used for synchronisation provides tighter synchronisation whereas longer time strips (e.g. start and end points of a recording) result higher slack across synchronisation points.

4.7. Summary

The UbiITS framework presented in this chapter covers all the technical, architectural and deployment details. The description initially provided the reference architecture and definitions of components that could be used as building blocks in constructing a DCIS. Subsequently techniques used for deploying the system architecture, event handling and temporal synchronisation have been described. Major components and techniques included in the UbiITS framework are listed in Table 4.3. Fig.4.19 indicates a process suggested for designing a system implementing UbiITS framework.

Functionality	Component/Techniques - Classes	Section Ref#
Reference Architecture	-	4.2
Abstract Devices	Unified Interface	4.3.2
	State Variables	4.3.3
Abstract Data	Event Data Type	4.3.4
	Stream Data Type	4.3.4
Data Flow Tools	Asynchronous FIFO	4.3.6
	Data Source	4.3.5
	Data Sink	4.3.5
Deployment Configurations	Stand alone	4.4.1
	Distributed	4.4.3
Asynchronous	Asynchronous Messaging	4.4.2
	Event Handling	4.5
	Distributed Events	4.5
Synchronisation Techniques	Time Stamping	4.6.1
	Clock Synchronisation	4.6.2
	Automatic Latency measurement	4.6.4
	Runtime Sample Monitoring	4.6.6

Table 4.3 List of components and techniques included in the UbiITS framework.

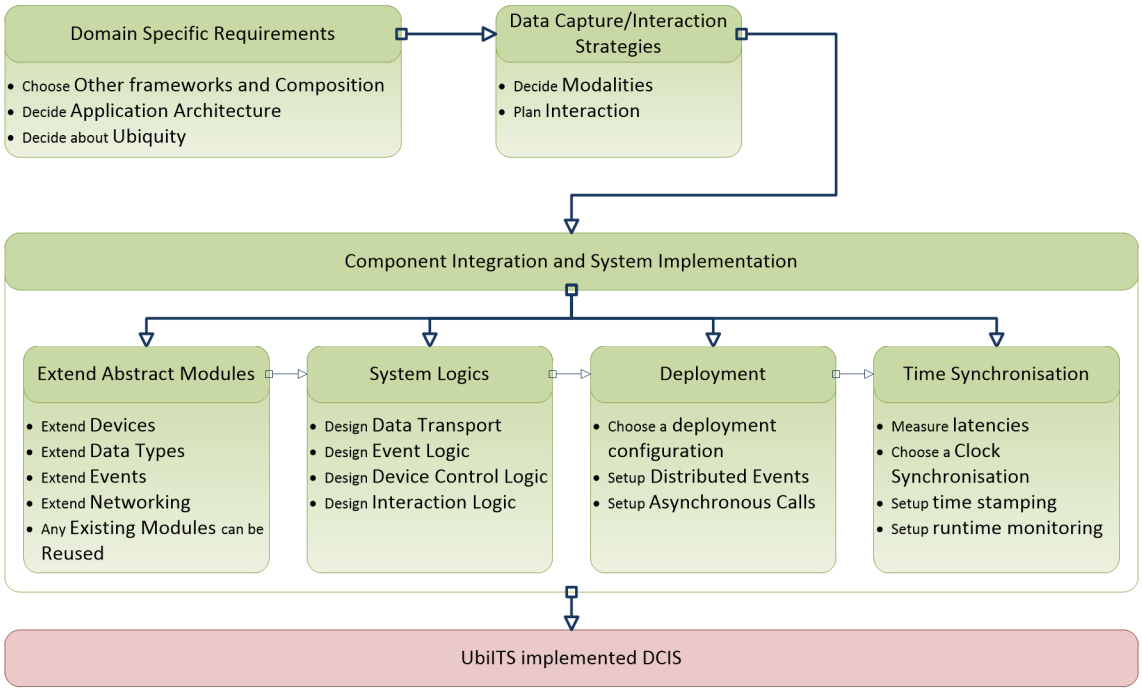


Fig.4.19 A design process suggested for implementing a system with UbiITS framework

The framework definition has been presented in a generic form, without referring to any solid implementation examples. Maintaining its generic form is imperative so that it enables the framework to comply with its ubiquitous, generic, abstract and extendable characteristics. Application case studies provided in the following chapters will present concrete implementation examples of the DCIS, justifying and demonstrating the promised functionality of the framework. A characteristic metrics on the framework will be established after the case studies as a summary evaluation.

Chapter 5. The application of ubiquitous multimodal synchronous data capture in CAD

Related Publications

Sivanathan, Aparajithan, Theodore Lim, James Ritchie, Raymond Sung, Zoe Kosmadoudi, and Ying Liu. "The Application of Ubiquitous Multimodal Synchronous Data Capture in CAD." *Computer-Aided Design*. November 1, 2013. doi:10.1016/j.cad.2013.10.001.

Liu, Y., J.M. Ritchie, T. Lim, Z. Kosmadoudi, A. Sivanathan, and R.C.W. Sung. "A Fuzzy Psycho-Physiological Approach to Enable the Understanding of an Engineer's Affect Status during CAD Activities." *Computer-Aided Design*. Accessed November 19, 2013. doi:10.1016/j.cad.2013.10.007.

Product design can be a complex and iterative process where the knowledge, experience and skills of a designer/engineer are articulated through the design tools. The reasoning given by the designer behind every action performed in a design activity is highly complex and comprises inferred information, explicit and tacit knowledge, emotive communication and transient exposures to the subject matter [156]. Despite the associated product constraints, a designer's knowledge and rationale play a significant role in the design. These are fuzzy by nature, difficult to capture and therefore difficult to document. Although self-reporting at each design phase can be applied, it is heavily dependent on the memory of the designer, open to interpretation and also creates interrupts during the design process itself.

Present-day computer aided design (CAD) solutions can produce a history of the designed solution within the internal bounds of their software package, e.g. revision history of a CAD part, tracking of changes made to a solution, etc. However they miss out the valuable information about external endeavours particularly those that happen during the design activity such as dialogues, sketches, notes, informative directives, knowledge and the emotional states of the designer, as well as other physical activities which influence the design process.

Literature lists a whole host of data required to be monitored and logged to enable design tasks in computer aided environments to be revisited, interrogated, analysed and understood [157], [158]. The future of design task analysis and knowledge/information capture lies in studies that involve the logging and analysis of formal computer aided engineering data e.g. product specifications, geometries, properties, calculations, simulations, in conjunction with the multimodal information about the activities performed during the design process. Various previous works reported about multimodal interaction capture involved capturing audio-

visuals, document access history, inter-personal interactions and psycho-physiological data in a time-phased manner [157], [159]–[161]. As a consequence having standard protocols would be highly beneficial, which researchers can use to implement such a monitoring methodology as a tool to take forward the next generation of design research.

This case study showcases the application of a generic framework for ubiquitous multimodal synchronous data capture, based around the capture of CAD system activities, which was employed to monitor and log a variety of inputs, interactions, psycho-physiological data and design solutions with a view to providing meta and chronological performance data for post task analysis. The framework predominantly addresses a technical solution for the ubiquitous and holistic capture of design activity. Through this work it has been demonstrated that it is possible to build up an end-to-end pipeline which ubiquitously captures the design activity and represents the captured meta-information against CAD records in a time-phased manner.

5.1. CAD - the case for unobtrusive ubiquitous design data capture

Jin and Ishino [159] proposed a design activity knowledge acquisition (DAKA) framework to extract the design activity knowledge by capturing the designers' design moves and used a function-based design operation-mining algorithm to extract meaningful design operation sequences. DAKA established the product model road map that represents the trajectory designers walked through during the design process by capturing the sequence of events occurring while interacting with CAD software.

Besides the CAD software and CAD models, a designer would also interact with other media such as text, audio, video and sketches. Shipman and McCall [160] proposed an integrated approach to capturing design rationale and associated design communications by means of the 'hypermedia'. Two Systems, PHIDIAS and Hyper Object Substrate (HOS) were used to capture and integrate a variety of hypermedia related to the design process and to structure the unstructured hypermedia information. A vector-based node-link structure was used to represent the connectivity between hypermedia, serving as the basis for design rationale. Although the structuring of hypermedia is studied, no temporal records were utilized in this work to relate them with the chronological performance of the design process.

Design activities are dynamic and any information resources accessed during a design session can be captured and potentially reused in future design tasks. However, structuring design knowledge, e.g. inferred, tacit, explicit and transitive, in a machine accessible format for storage and retrieval for computer-based knowledge support is not trivial. Campbell et al. [161], [162] explored a methodology for profiling computer-based design activities relative to the order and timing of resources accessed during a design session, where the data is captured

along with the creation of formal design representations, e.g. design drawings, reports, etc. based on Bayesian inference. They then tried to relate it to the designer's focus or goal which was interpreted by the actions observed at the computer interface, e.g. keyboard input and interaction with the CAD software, so that a relationship between documents based on the context of their usage can be identified.

The Knowledge Enhanced Notes (KEN) system introduced by Conway et al. captures collaborative activities in a design meeting and generates an enhanced documented record of the meeting with a view of re-using at later stages in the product lifecycle. In addition to the discussions and decisions made during the meeting, KEN records the resources accessed and the temporal information about them [157]. The benefits of the multimodal information has been generally realised specifically as informal data about the design process in addition to formal design records, which enables revisiting and understanding the design process in product's extended lifecycles [163], [164].

Revisiting and interrogating the design process with regards to the multimodal metadata requires a pristine link between the formal data and informal data. Generally, this has been overlooked and the informal data searchable or not-searchable on its own will not have the associativity with the formal data. Therefore the captured metadata should be associated to the part name or a product lifecycle management (PLM) identification property in order to provide the bidirectional accessibility. Preferably, the best time to capture this link is during the actual activity when the formal design records are being created or accessed and the informal data is being captured.

Analogous to CAD, Rea et al. used BAMZOOKi, an intuitive 3D design environment to study design activities [165]. This work specifically enabled the automatic capture of the design process via a customised BAMZOOKi interface and generated log files based on extensible mark-up language (XML) and Process Specification Language (PSL) during the design activity. The log files were subsequently processed to output various representations of design knowledge. Integrated computer-aided manufacturing DEFinitions IDEF0 [166] and Design Rationale Editor DRed [167] diagrams were automatically generated to formally represent the designer's activity and design rationale. English-syntax instructions, annotated video clips and a design storyboard were generated to represent the design knowledge and processes. Similarly design activities in a virtual reality based cable organisation system were also studied by capturing user activities identical to the CAD design process [168], [169]. This further demonstrates and validates that design activities can be potentially transformed into understandable and CAD-neutral format knowledge representations.

5.2. Capturing user activity

Comparable to analysing a CAD activity, capturing user activities is common in Human Computer Interaction (HCI) studies. Usability studies for investigating software application user interfaces (UI) extensively used user activity monitoring systems [31], [32], [170]. A survey by Hilbert and Redmiles [171] classified and described a wide range of techniques used in HCI research for extracting usability information from user interfaces. They categorised which usability indicators were analysed, e.g. user behaviour, performance, cognition, attitude, stress level, and which multimodal data collection techniques such as UI events, audio, video, psychophysical recording and subjective measures such as interviews and surveys, were used to study the usability. This work also highlighted that user activities can be classified and analysed into different levels according to their level of granularity and abstraction; the abstracted activities can be inferred by interpreting a stream of events related to the low-level interactions [171].

Fig.5.1 illustrates examples of CAD user activities during a typical CAD task, approximately representing the variation from low-level interactions to inferred abstract tasks. A balance between the granularity of the data captured and the level of inference needed to incorporate the abstraction should be maintained while capturing and analysing user activities [172]. Accordingly, adequate low-level, granular activity data should be captured to recognize reasonably abstract tasks [170]. Too much data, when combined with extraneous information, creates difficulties in drawing inferences about the task and can demand high computational and storage resources.

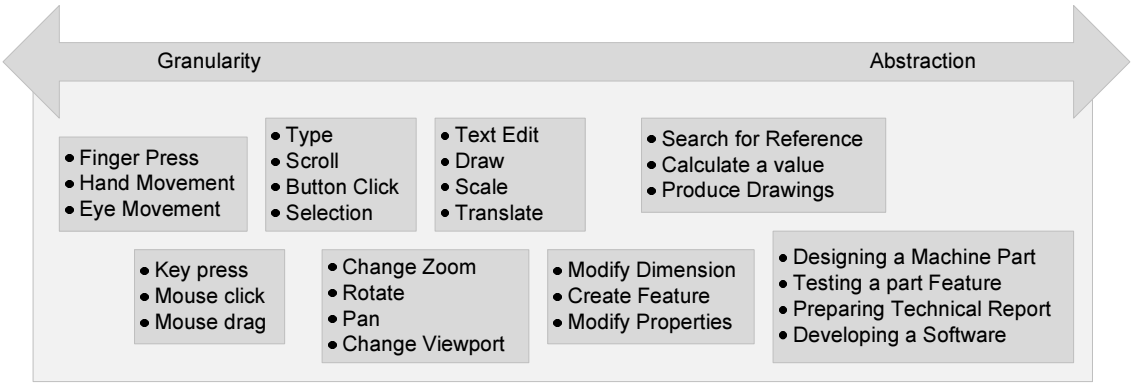


Fig.5.1 Example activities in a CAD task approximately illustrating the granularity of the user activity and the abstraction of tasks

Standard operations in a PC, such as file operations, text manipulation, accessing the web, voice calls, etc., are often monitored to analyse the user activity [31]. Capturing events automatically generated by applications, instead of low-level keyboard and mouse

interactions, is a commonly used technique in user activity monitoring systems. These events are usually pre-determined and the applications customised to automatically generate these events while the user interacts with the system [172], [173]. Events are chosen and organised based around the domain of study, for example a study about document interactions considers file system events, internet browser events and application window events [162].

Software tools are normally required to capture and exchange event information across the UI application and monitoring software. Microsoft component object models (COM) and dynamically linked libraries (DLL) are often used to build the infrastructure for receiving event notifications in Microsoft applications [31], [172], [173]. Intercepting messages invoked by the operating system for keyboard and mouse events is another commonly used technique for monitoring user activity [172], [174].

Events raised by the application or the user interface represent instantaneous activities occurring for a discrete period of time. In contrast, user activities can be observed in a continuous manner, e.g. video and audio recording, screen capture and continuous mouse movements. [32], [33]. Psycho-physiological measures are increasingly being employed to establish a better understanding of the internal state of the user. Heart rate/electrocardiography (EKG), facial muscle activities, eye tracking, pupillometry, electroencephalography (EEG), electromyography (EMG) and galvanic skin response (GSR) are some of the commonly used biometric measures to monitor user activity [32], [34], [35], [175]–[177].

User activity monitoring systems have largely focused on logging events occurring between the user and the UI. However, recording user activity in a task-based situation creates more challenges. A design task involves human factors beyond examining the interaction solely between a designer and software. Personal aspects such as knowledge, experience, creativity and emotional perception do influence the design solution. Consequently, psycho-physiological data provides an alternative for analysing affective design activities. Complementary to UI event logging, which provides an explicit form of information about the user interaction within the UI, the human-centric data represents an implicit form of user interaction.

A need for a common set of tools to integrate and build up a user activity monitoring system has been noticeably identified in the past [2]; however, this work is the first attempt at specifying, designing and implementing such a generic framework for CAD.

5.3. Requirements for a ubiquitous framework

Sections 5.1 and 5.2 has established the case for using informal metadata relating to the design process in addition to formal design records has been described and consequently the requirement of multimodal capture. The design process necessitates a ubiquitous and dynamic system that dictates that any capture tools are unobtrusively embedded into the working environment; it must not cause interruptions of or add extra workload to the design process. The multiple modalities also necessitate tight temporal synchronisation tolerances between captured data streams in order that a meaningful understanding of the CAD activity relationships can be established. For these reasons systematic, implementable arrangements are presented to accommodate a wide range of data capture tools that blend seamlessly into the environment and the process. It particularly addresses a technical solution for consistently extendable, ubiquitous metadata capture systems.

5.4. Experimental setup

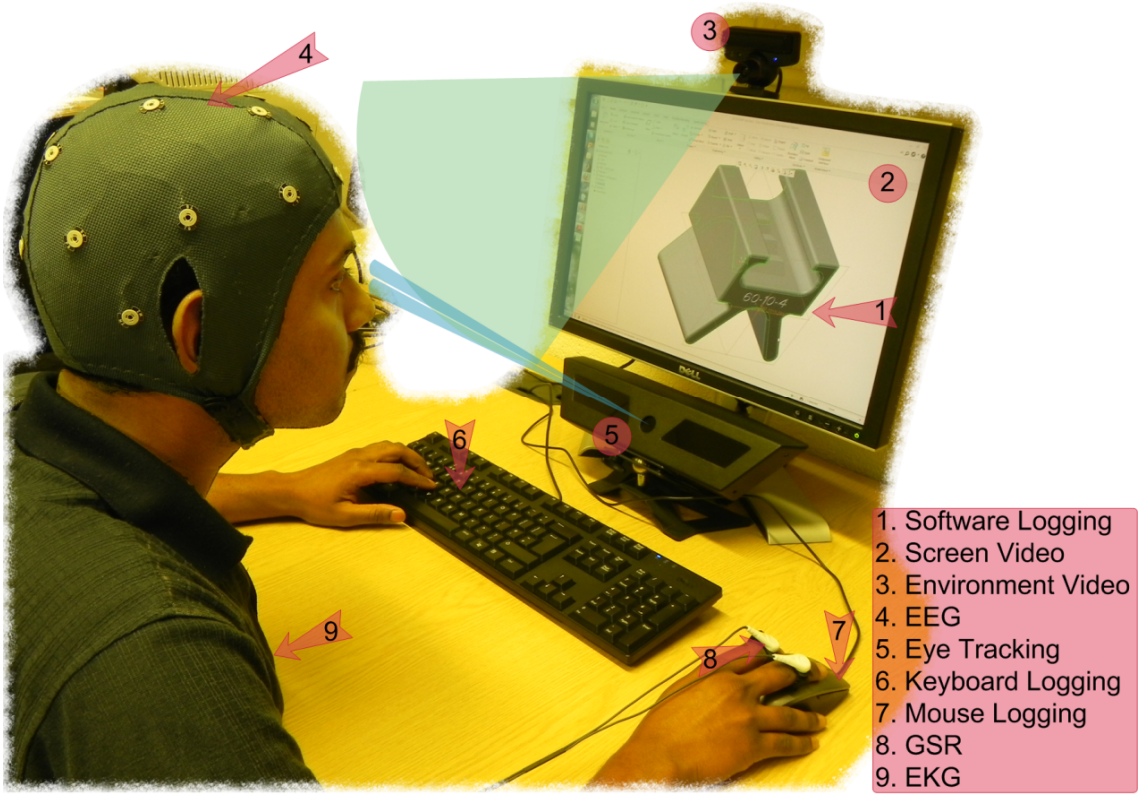


Fig.5.2 Ubiquitous multimodal data capture framework used in a CAD design environment. Multiple capture devices used in the system are listed and highlighted.

An experimental ubiquitous DCIS was setup to demonstrate the applicability of the UbiITS framework to an engineering design task (Fig.5.2). This experimental system was also used to test how its components fulfil the requirements for capturing an engineering design activity. Previously described concepts and modules of the framework are implemented and tested by

applying them to the experimental capture of design activity and thereafter justified by examining the captured data during the experiment. An outline of the experimental design task followed by descriptions of the framework implementation and how its modules were used to handle issues related to the design environment will now be detailed.

A design task to optimize a bracket to support the given pin force was to be trialled on an industry standard CAD package, i.e. Siemens NX™. A total of 24 engineering students were recruited to establish a record on their cognitive loads during design activities. There was no time restriction for the user to complete the task. The design task was divided into two stages:

1. To modify the bracket such that it will not fail given an applied pin force. The student was required to calculate the allowable tensile and shear stress based on the bracket material and then, through the interface, configure the cross-section (breadth and thickness) of the bracket (Fig.5.3).

2. To calculate the tensile and shear stress including the weld type and sizing for the bracket.

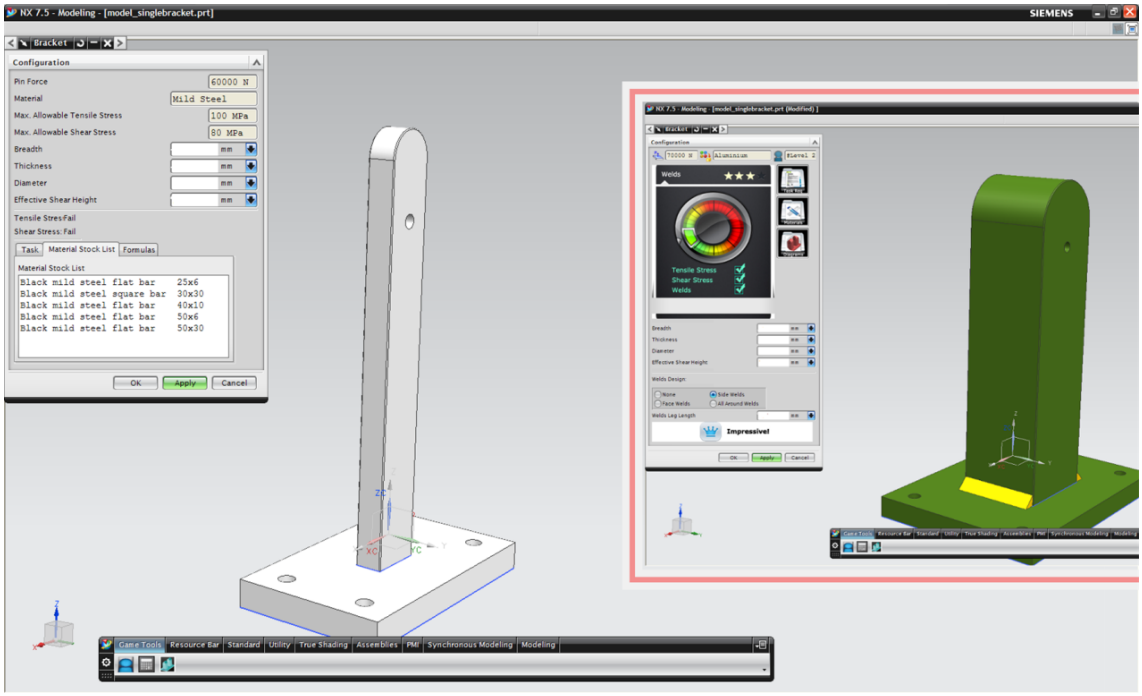


Fig.5.3 Illustrates the experimental task in the customised NX interface which allows performing the stress calculations and shows the subsequent results

In order to understand the affective states and the cognitive processes of the designer during the iterations of the given task, psycho-physiological measurements were recorded and synchronised with other modalities such as keyboard and mouse interactions, video recording, etc.

High temporal synchronization was required in the recorded data, which included the EEG, EKG, eye movement, NX log file, mouse clicks and a video of user design activities. The EEG and ECG data were processed for affect artefacts and pupillary measures associated to the cognitive load. By studying the neuro-psycho-physiological attributes associated to a CAD-based engineering design process, it is envisaged that interrelated engineering behaviours can be more deeply understood and consequently, lead to more natural and intuitive CAD user interfaces.

5.5. Employing UbiITS framework to solve technical problems

Each of the capture tools within the ubiquitous capture system produces a variety of data types depending on the captured content, e.g. discrete key press, mouse movement coordinates, video, location-based multichannel physiological measurement such as EEG, etc. The characteristics of these multimodal data vary extensively based on the format of the data, its resolution and frequency. Care should be taken to handle those dissimilar data types, for example, the way the 19 channel EEG data is treated, which corresponds to different locations on the head, is different than, say, 2-dimensional pixel-based screen capture; data buffers involved in retrieving the data and the processing algorithms used also vary.

In addition to the dissimilarity in data formats, the method by which they are captured is also important. Several data channels are embedded within a stream i.e. continuously at a constant frequency, e.g. EEG, eye tracking, video, while the others are captured as discrete events occurring in the environment, e.g. CAD activity logging, key presses. Remarkably, some of the data can be captured in both streams and in an event-based manner, e.g. mouse activity can be monitored for discrete clicks while tracking the curser movement continuously and tracking the eyes continuously while triggering specific events when the user looks at a specific region of interest on the screen.

5.5.1. Bandwidth and computational load

The bandwidth and frequency of a data channel are important factors in deciding how it should be handled. Biophysical signals, e.g. EEG, GSR, are captured at high frequency (2048Hz) from USB hardware. There is the potential for missing data packets from devices if it were not read at the required frequency. Also crucial is the allocation of appropriate buffer sizes and to remove the data from the buffers at an adequate rate. HD video capture needs handling vast amounts of data (1280x720 pixels x 3 byte colour depth X 30fps = 79.10Mbytes per second) and potentially creates a bandwidth bottleneck within the data capture system. It is common to use a video codec to compress the video in real time; on the other hand runtime video compression puts high demand on the CPU. Although CAD logging is normally not a

computationally demanding task, the events triggered by the CAD software are sporadic since it is based upon user activity, e.g. the user spins the model, changes the view zoom occasionally. This produces varying amounts of data in short but intensive bursts, producing a non-uniform load on computational resources which might cause issues in the real-time scheduling of tasks involved in the system.

5.5.2. Online data exchange

Data handling modules implemented in the UbiITS framework were used to address issues related to multimodal data. Abstract and modular-based definitions of stream type and event type data built in the framework were extended to support each data modality. Fig.5.4 shows example configurations of how the data handling units in the ubiquitous data capture environment were linked using the appropriate modules. Brief descriptions of each configuration are now given.

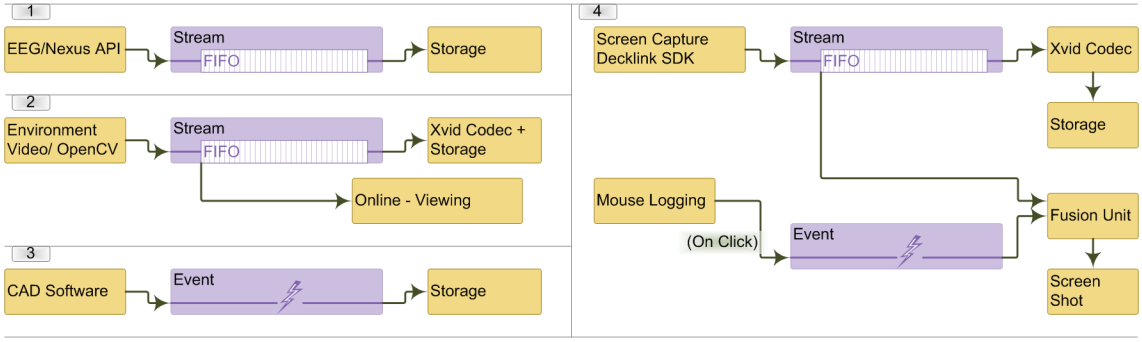


Fig.5.4 Four examples of data flow configurations for stream and event-based data

First in first out (FIFO) buffers provided by the UbiITS framework were extensively used for sharing data from one unit to the other, where FIFO buffers supports exchanging data between multiple execution threads. Multi-threading provides better handling between time critical tasks and long running background tasks. For instance, collecting data from EEG hardware is a time critical high priority task. Therefore a high priority, high frequency task can collect EEG samples from the hardware and store it in the FIFO, whereas a long running storage task can transfer the data in batches from the FIFO to a disk storage. The abstract stream data module implemented in the framework was extended to support EEG data formats. Correspondingly, the same module was extended to support other stream-based data types in the following configurations.

5.5.3. Online viewing of captured data

It is reasonable to have the data capture system with real-time viewing capability which can be used for inspecting the captured data online. The video capturing the user activity could be

viewed online while the recorded video frames are compressed by the codec and stored in a video file. However, online viewing is only a supplementary feature and has low importance compared other real-time tasks. It is good enough to miss video frames for viewing and hence appropriate to run a low frequency, low priority thread with flexible timing requirements. FIFO modules provided by the UbiITS framework incorporated a feature to retrieve samples without removing the samples from the buffer, so that it allowed every frame to be securely collected by the storage thread.

5.5.4. Handling CAD system events

In contrast to the stream module, an event based module is implemented in the framework which deals in particular with sporadic data. Events triggered in CAD software are passed through this event module and then linked to the other system units as necessary. A fusion unit was employed to retrieve a frame from the user screen video stream whenever a mouse click event is received. Two functions, namely capturing the user screen video and generating a screen shot when a mouse click event is fired, are combined in this configuration.

Data handling modules implemented in the framework are designed to be lightweight and high performance. In contrast to other data logging systems [54] which upload the data to a centralized database, data is saved by the framework in local files in a distributed manner, therefore enabling high bandwidth and low latency data handling. Nevertheless, the captured data can be committed in batch operations to a central repository using a low priority task if required.

5.5.5. Integrating multimodal capture tools devices with a centralised control panel

A centralised control panel is required to control all the capture devices or tools, where every device can be accessed from that central point allowing to start/stop capturing the CAD station activity. Two possibilities in this central control point are that the whole capture process can be manually controlled (i.e. CAD engineer or a facilitator) or attached to a fully automatic mechanism, for example the capture starts/stops along with the CAD software and controlled by a customised interface panel within the CAD software. Every device interface and corresponding API/SDK was encapsulated within a wrapper interface that is compatible with the unified access interface defined by the UbiITS framework. Fig.5.5 illustrates an example unified access interface and centralized control panel.

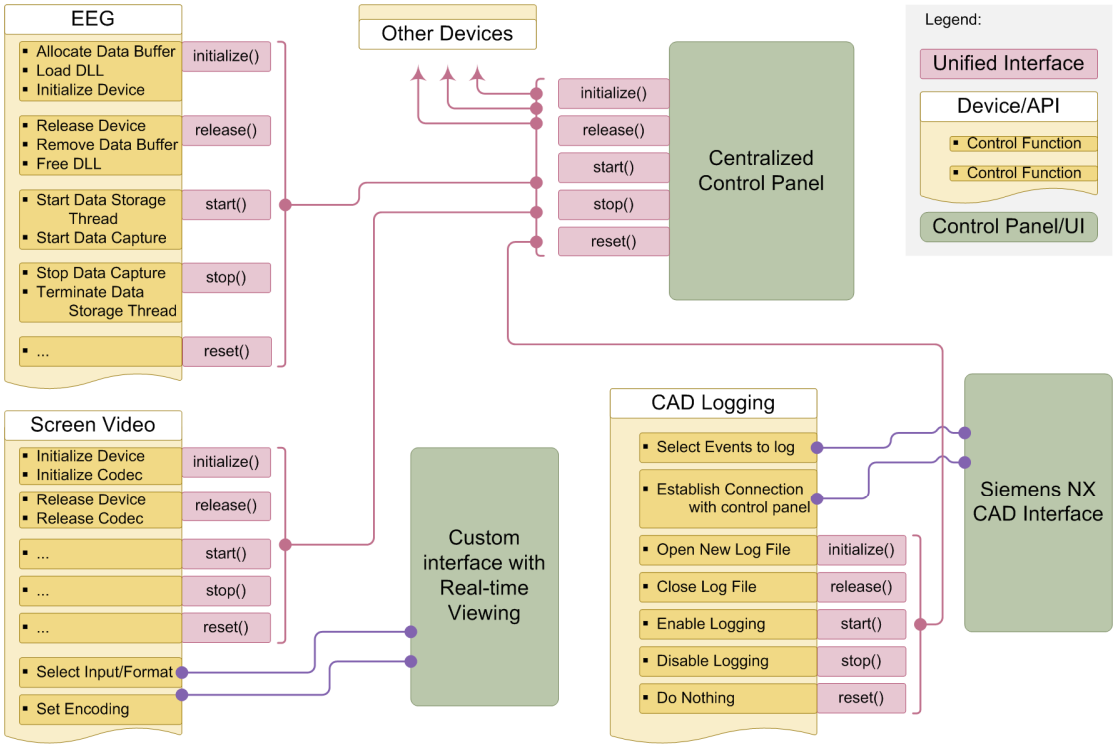


Fig.5.5 An illustration of a unified interface with centralised and device specific control panels. A part of the system and example functionalities are used to describe the concept.

5.5.6. Interconnection of distributed devices

The diversity of devices used in the system for monitoring CAD environments often requires special considerations in choosing hardware and software platforms. Support is provided by various device vendors for different operating system platforms and maintaining issues caused by legacy software/hardware influences selecting the hardware and software platforms. Incompatibility between devices and platforms occasionally requires a mix of platforms. UbiITS supports interconnecting devices running independently in isolated platforms with the aid of a communication interlink, e.g. TCP, UART Messages, that relay the unified interface function calls and also the synchronisation related messages. Furthermore when the data streams or an events are exchanged between components of the framework, the latency in these interlinks also needs to be counted when synchronisation is considered.

5.5.7. Integration of ubiquitous devices in the system

The Siemens NX™ CAD software was customised using the UniGraphics/NX Open API according to the unified interface structure as proposed in the framework. The customised module also generated markers for predefined actions, e.g. modifying the values of text boxes, pressing command buttons, pan and zoom -view manipulations, etc., carried out in the CAD software. This module was also responsible for regulating the other capture tools setup in the

environment. For example metadata capture of other devices is started/stopped when the designer starts/stops working on the CAD, i.e. controlling the whole metadata environment.

Fig.5.6 shows how the capture devices were setup and how three different workstation platforms were used in the ubiquitous data capture environment for the CAD task. A low latency system, PC1 (Windows 7, 32bit), is used as the centralized control panel and for time stamping purposes. Simultaneously, it is also used with EEG and eye tracking devices, for which platform drivers and APIs only support 32bit platforms. PC2 (Windows 7, 64 bit) has been purposely tuned for high bandwidth and storage capabilities with RAID-0 disk storage and USB3 features. The CAD workstation (Windows XP, 32 bit) was built addressing the legacy support issues. TCP interlinks were implemented to interconnect distributed devices with the centralized control panel and to create connections between devices wherever the data flow between different distributed device units was required.

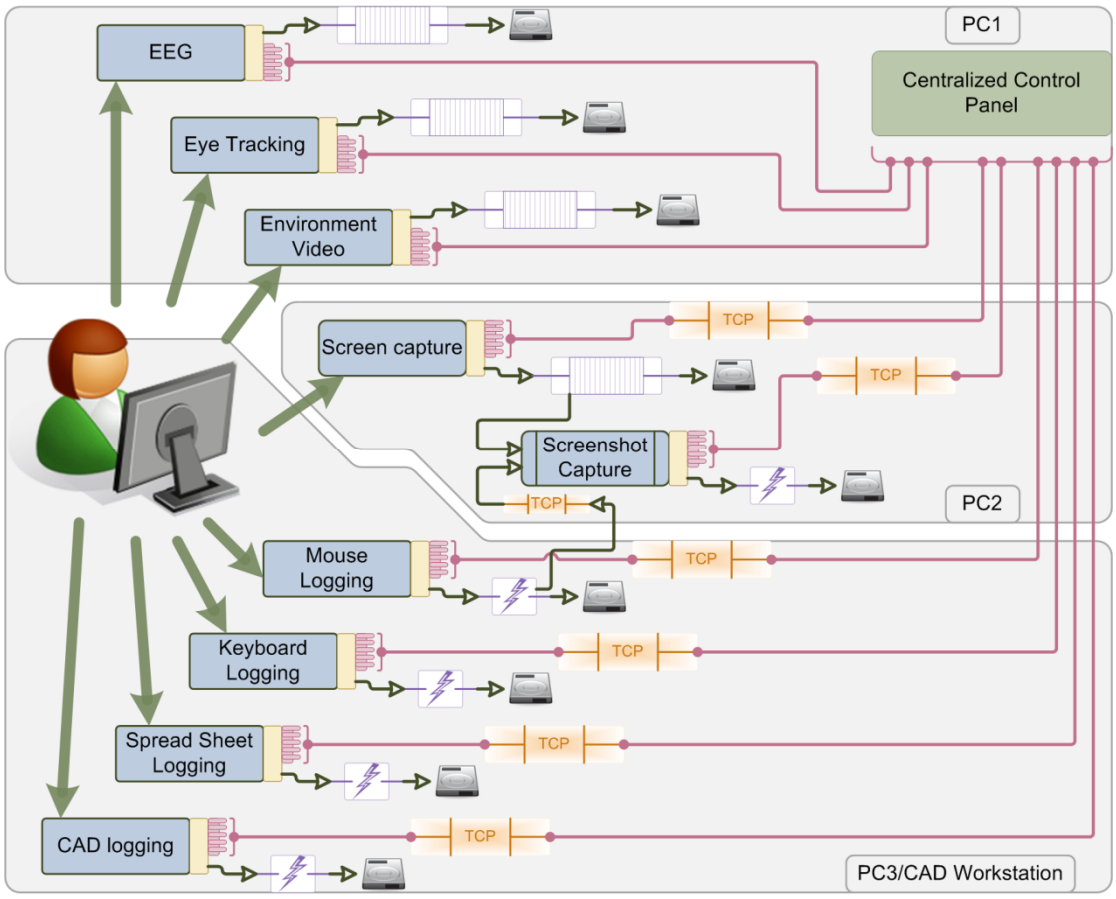


Fig.5.6 Arrangement of device units showing how event and stream modules are used to handle the data flow and the unified interface is used to access device functions. TCP interlinks have been created to support the interconnection of distributed devices.

5.5.8. Data stream synchronisation

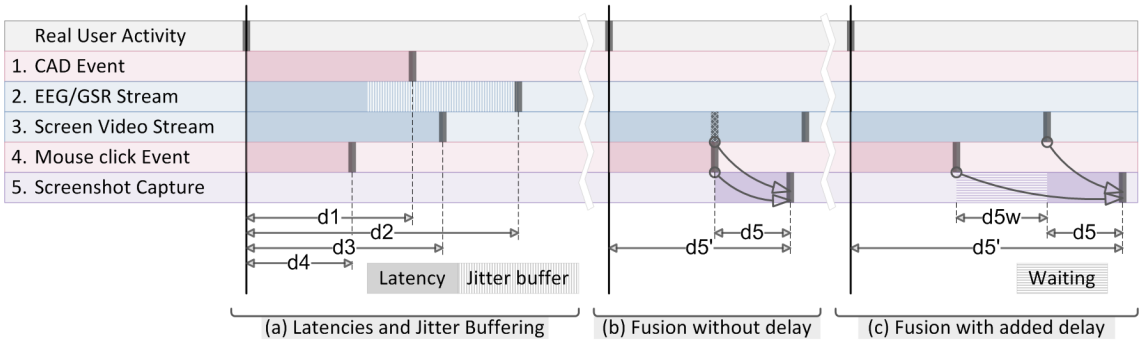


*Fig.5.7 Illustration of latencies in data channels with respect to instantaneous user activity **(a)**. Fusion units, i.e. screenshot capture on mouse clicks, are influenced by latencies. Offsets in the streams create skew **(b)** in the generated screenshots. This can be compensated by adding an extra delay **(c)** as required. **d1-d5** = latencies in channels, **d5w** = added waiting time, **d5'** = overall latency for the screenshot.*

Data samples collected via miscellaneous devices are routed through various data paths thus possessing disparate latencies. These streams must be aligned to produce valid correlations and hence usable information can be generated. The validity of the inferred data produced by fusion units depends on the temporal alignment of the data fed into these data fusions. For example, the screen video, eye tracking and GSR must be temporally synchronised to derive a reasonable conclusion about user's emotions in an instant [178]. It is also important to synchronise the events occurring in the CAD environment, e.g. events triggered by the CAD software, mouse clicks, with the data streams such as biophysical signals and video. An illustration of the potential latencies involved in data channels is shown in Fig.5.7. To address this latency/offset compensation parameters were introduced for every data channel (see: section 4.6.3). The offset parameter is used in online-fusion units to compensate for the offsets in fused channels or in post-capture analysis to re-synchronise the channels. Configuration 4 in Fig.5.4 is the fusion unit used to capture screenshots from mouse clicks, where the screenshots are skewed by the difference in offsets in the data streams (Fig.5.7, b and c).

5.5.9. Handling latency and jitter in data channels

Mouse clicks and screen video stream are captured by different devices and associated APIs. The offset between these channels creates an observable skew when these channels are fused together. The video frames and mouse clicks were analysed to validate synchronisation and the method of latency/offset compensation. Although the offset compensation was applied for every channel, these two channels are chosen for analysis because the mouse pointer on the screen and the button click event can be manually observed in the video frames. Mouse x and

y coordinates logged in a click event by the Windows hooks is compared with the screen video frames. Fig.5.8 shows the timeline of video frames, a mouse click event and the mouse click coordinates. The skew can be visually observed where the click coordinates and the mouse pointer do not match. When the offset compensation is applied the mouse pointer is then aligned with the logged coordinates in a subsequent frame, hence validating the synchronisation.

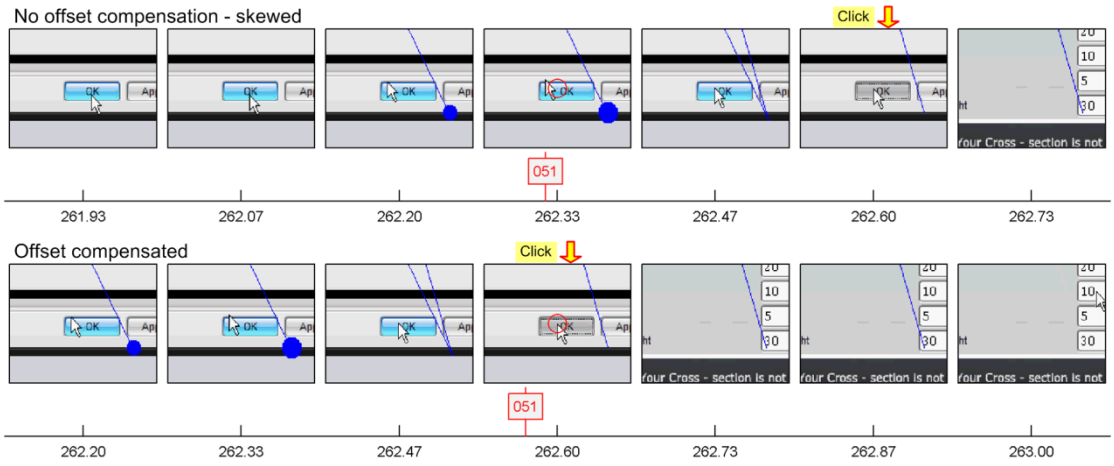


Fig.5.8 Timeline of grabbed frames and mouse click event, where the logged mouse coordinates on a click event is superimposed (red) on the closest video frame. The skew can be visibly observed (top) before compensating for the offset and thereafter the click is synchronised (bottom) with the video stream after applying a compensation of 267ms. Eye-gaze points are also overlaid on the frames (blue)

Monitoring the jitter and frame rate of a device at run-time provides more information about the runtime behaviour of the device and data stream generated by the device. This information is captured by the data handling modules implemented in the framework (Section 5.5.8) to provide a precise understanding about the dynamic characteristics of the synchronisation. For example, the Nexus bio-physical monitoring device samples signals at 2048Hz according to the manufacturer's specification. The number of samples delivered by this device via its API is observed at fixed time intervals. Subsequently the instantaneous sample rate was calculated using the time provided by a high precision hardware clock in the central processing unit (CPU). The continuously measured sample rate and the observation of jitter are shown in Fig.5.9. The presence of jitter and the deviation in the sample rate in this data stream have been revealed using the data handing modules implemented in the framework. Similar distortions were also observed in other devices (e.g. eye tracker, frame grabber), which were possibly caused by the non-deterministic characteristics of the data paths due to USB, Ethernet and the inaccuracies in task scheduling. Consequently, FIFO buffers

were implemented in the framework to treat the jitter and offset compensations were applied to individual data channels so that they can be temporally aligned.

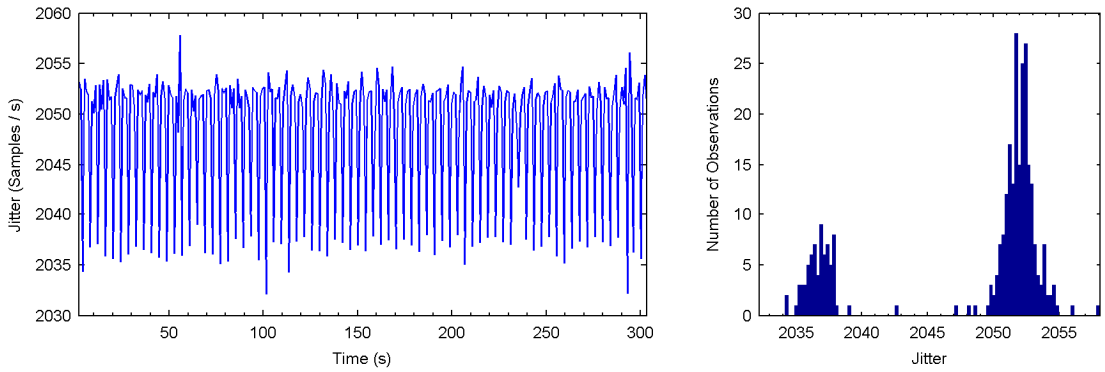


Fig.5.9 Jitter observed in run-time for the bio-physical monitoring device, with the theoretical frequency of 2048Hz. The number of collected samples has been recorded every second and the resultant sampling rate is calculated against the CPU's hardware high precision clock.

5.6. Bio-physical signals synchronised with CAD Events

Events triggered by the CAD software while the engineer performs the design task are logged by the customised Siemens NX interface. These events were logged in a log file (Fig.5.10) with a distinctive event identifier and the corresponding time such that it can be synchronised and correlated with other data channels. These event identifiers are used by event handling modules in the framework to handle them unambiguously.

```
!!!&MACRO EVENT FOCUS_IN 1 0, 1900544, 0, 0, 0! (Application CB Nested)
&MACRO EVENT FOCUS_IN 1 0, 1900544, 0, 0, 0! Task Requirements

#<EVT MUSCLK0055 0073511055646363>#
&MACRO EVENT FOCUS_IN 1 0, 3145731, 0, 0, 0! Breadth
&MACRO A                1 31457      RN 0) = "20"  ! Breadth
                Event ID        Time

#<EVT MUSCLK0056 0073511055649843>#
&MACRO MODIFY_VERIFY_EVENT 1 0 3145731 1 STRN 0 2 insert = 4 proposed = 4 current = 20  ! Breadth
&MACRO EVENT VALUE_CHANGED 1 0, 3145731, 0, 0, 0! Breadth
&MACRO ASK_ITEM 3145731 (1 STRN 0) = "4"  ! Breadth
&MACRO MODIFY_VERIFY_EVENT 1 0 3145731 1 STRN 1 1 insert = 0 proposed = 40 current = 4  ! Breadth

#<EVT MUSCLK0057 0073511055654492>#
&MACRO MODIFY_VERIFY_EVENT 1 0 3145731 1 STRN 1 1 insert = 0 proposed = 40 current = 4  ! Breadth
&MACRO EVENT VALUE_CHANGED 1 0, 3145731, 0, 0, 0! Breadth
&MACRO ASK_ITEM 3145731 (1 STRN 0) = "40"  ! Breadth
&MACRO EVENT FOCUS_OUT 1 0, 3145731, 0, 0, 0! Breadth
```

Fig.5.10 A section of the CAD log file is shown with the event ID corresponding to a mouse click event and time.

A timeline of events triggered in the design environment and bio-physical signals are shown in Fig.5.11. 19 channels of EEG signals and EKG were captured using the Nexus device while eye gaze and pupil radius were captured by the Tobii eye tracker. These independently running

devices and the CAD logging were integrated using the framework to function as a ubiquitous capture system.

The eye blinks captured in the data were also used to validate the temporal alignment of integrated data streams. Eye blinks can be easily identified in the EEG data stream by the eye blink artefacts, even when the eye tracker loses tracking of the eyes when the user blinks. Fig.5.11 shows a tight temporal alignment of these data streams where EEG and eye tracking data can be visually examined for the blinks. The results clearly demonstrate the implemented data handling modules and synchronisation techniques of the framework.

5.7. Design activity representations

The time-phased log files recorded during the design activity were parsed by automatic tools [179] to produce various design activity representations. A modified DRed representation that incorporates the time duration of tasks is illustrated in Fig.5.12. Time-phased captured EEG signals were automatically processed for emotions using a fuzzy model [180] and synchronised to the CAD events with the intention of analysing the engineers emotions during the design task. Fig.5.13 shows the resulting processed emotion data synchronised and mapped onto an IDEF0 diagram in relation to the events associated with formal design decisions.

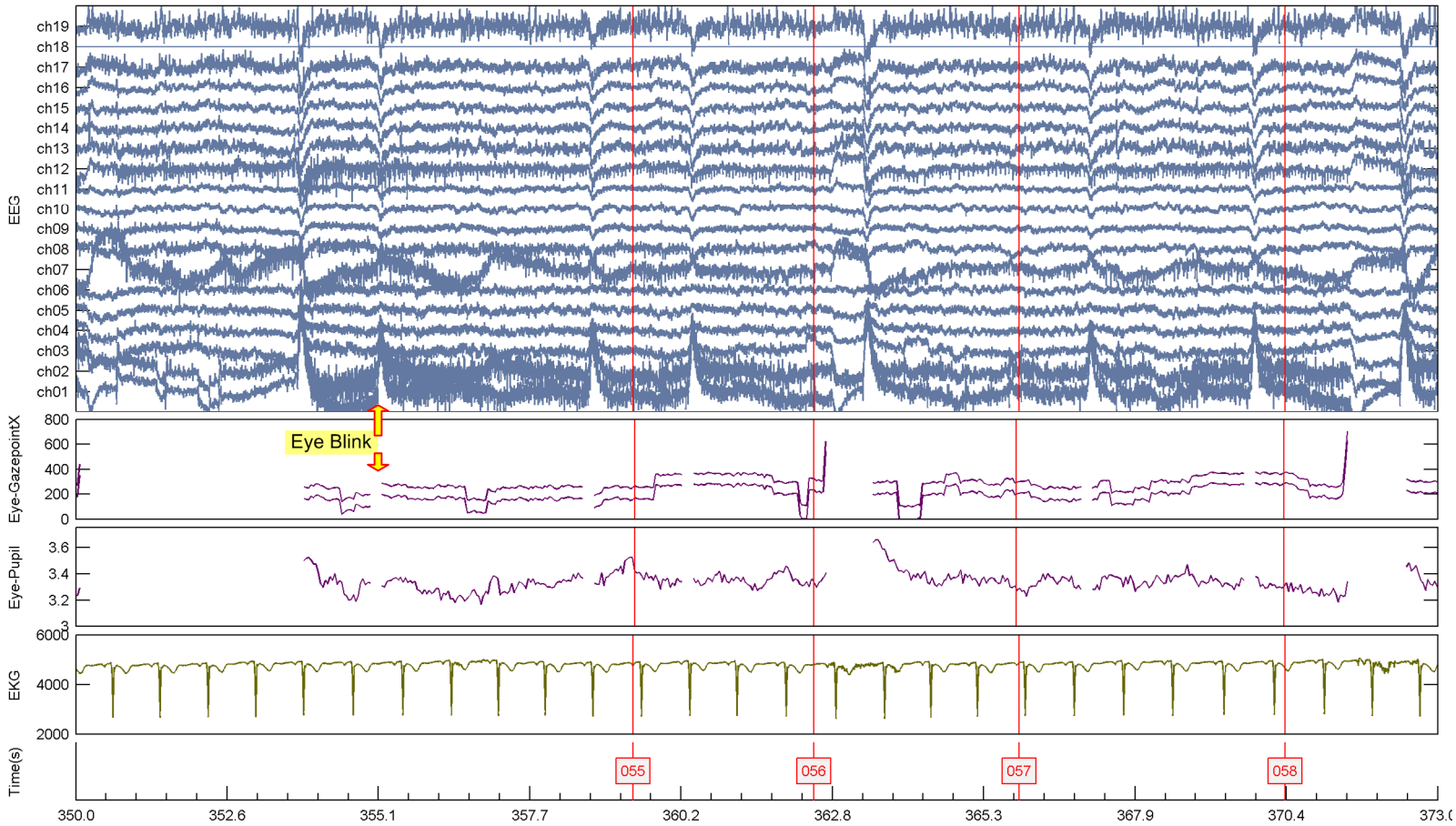


Fig.5.11 Events generated by the CAD interface have been synchronously inserted in the EEG, EKG streams captured by a bio-physical monitoring device and gaze point (left, right eyes) and pupil radius captured by an eye tracker. Events triggered by CAD software are indicated with unique event IDs. It can be visually examined that the eye blink – a physical action captured by two independent devices is accurately temporally aligned.

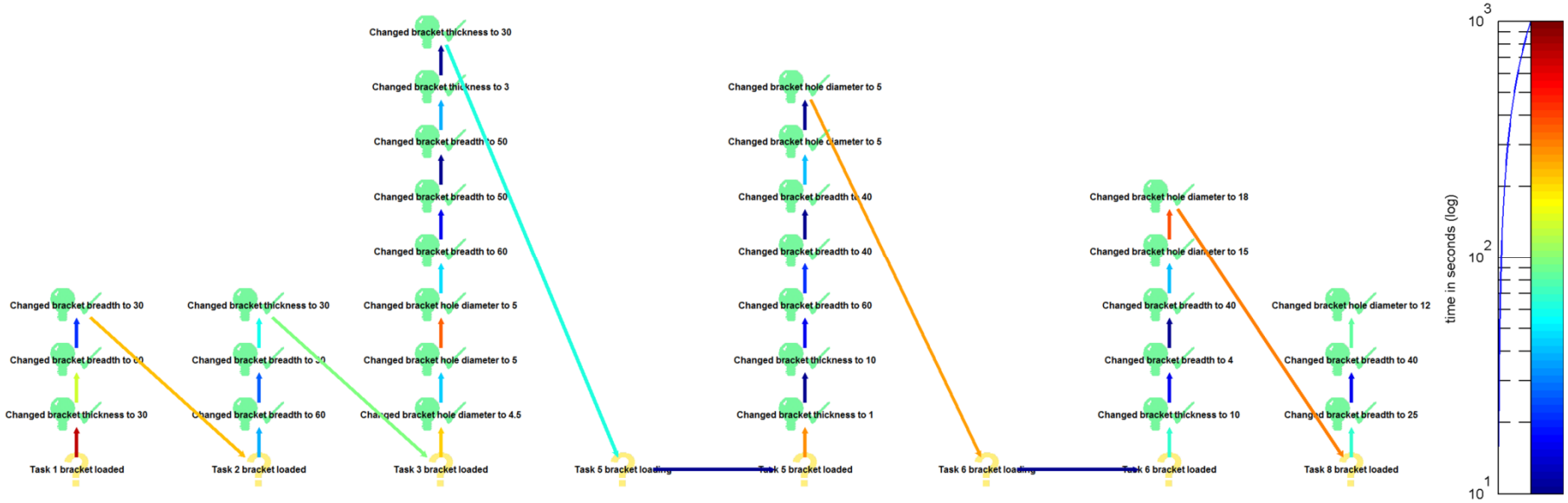


Fig.5.12 A DRed Diagram, automatically generated by post processing the synchronised log file, showing 8 activities performed modifying the bracket design in the CAD task. The accurate time duration of each task has been illustrated by the colour code. A logarithmic scale has been used to point up both short and prolonged tasks.

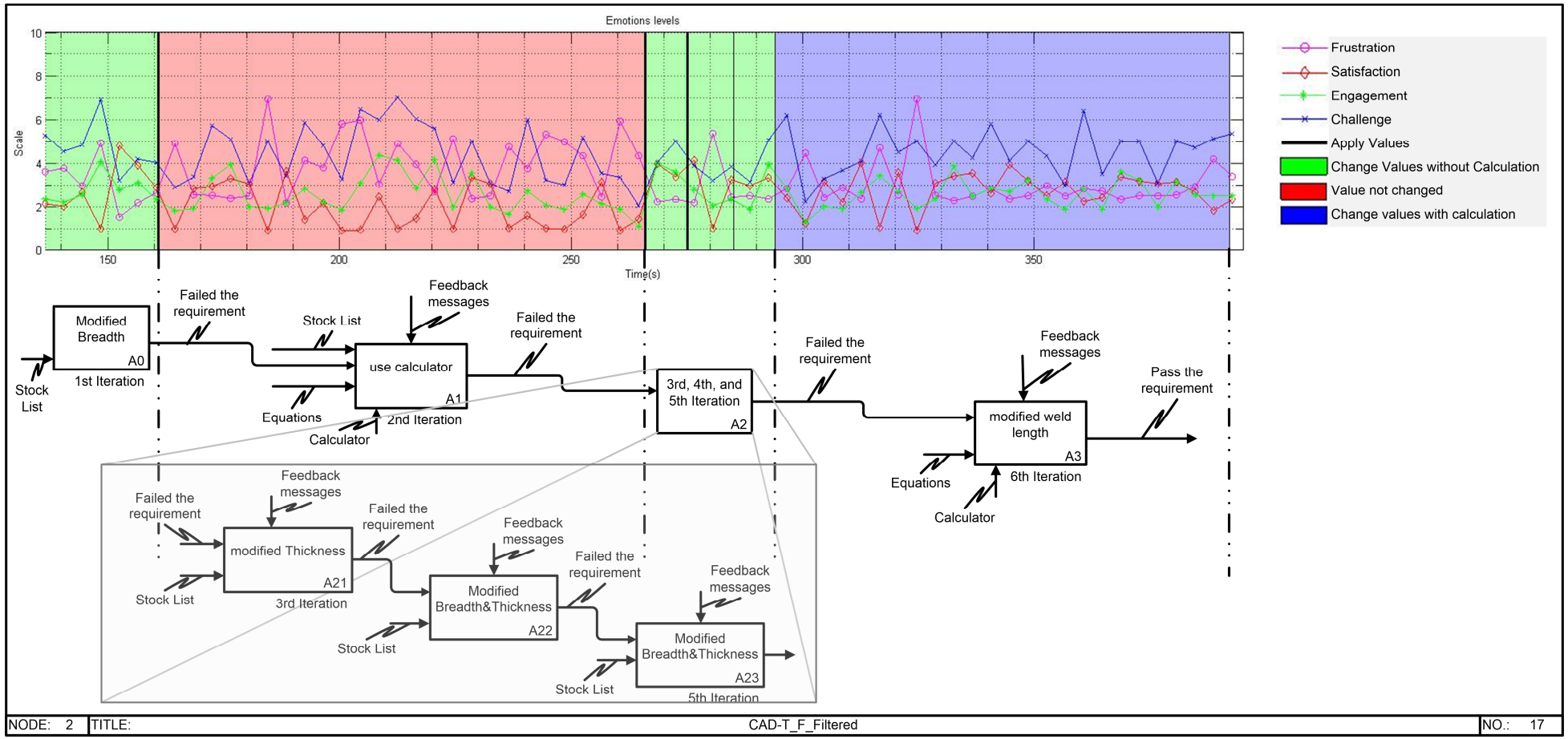


Fig.5.13 An IDEF0 diagram describing the iterations of participant while working on the Bracket Design Task, in conjunction with participant's emotional levels of frustration, satisfaction, challenge and engagement.

5.8. Outcomes of the ubiquitous meta data capture

Various studies were performed in the current state-of-the-art research with the aim of understanding the affective status of the user [181]–[185]. Employing the results from these studies in a ubiquitous CAD environment with multimodal signals is far from trivial. There are many technical challenges to be overcome before building up such a complicated system and therefore this study centred on the technical resolution of multimodal ubiquitous data capture. Therefore the finer details of the psychophysiological data processing fall outside the scope of this study.

Synchronisation-related evidences were presented which exhibit the time and chronological event synchronisation aspects of the framework. IDEF0 and DRed illustrations were provided to demonstrate how the metadata could be mapped against the CAD task progress. These illustrations associated various stages and iterations of the design solution with the user state interpreted from EEG. In summary, the evaluation of the framework via this CAD task demonstrates the capability of the framework and its ability to capture metadata in an engineering design task. Further it has been shown that a standard CAD system can be extended to support sophisticated metadata capture, an application for which it was not originally designed.

5.9. Discussion – Reflections and opportunities

Engineering design is a complicated activity where designers gather information from various sources, perform computations and add value to the product by inputting their knowledge. Designers need to be supplied with sufficient information and rationale about the earlier designs and decisions so that they can make well informed decisions during product development. The tacit knowledge used by designers is valuable content which is also incorporated and implicit in the product. Capturing, transporting and formalising tacit knowledge across time (i.e. from previous designs into latest designs) or places (e.g. from one person/team to another, different stages of designs, etc.) is not straightforward. Synchronously capturing various inputs, interactions and biophysical data along with the formal design data demonstrated here is a potential solution to meet the challenge of facilitating the capture and transportation of performance data and knowledge.

The meta-data captured during the design activity provides new-levels of traceability of design knowledge, enabling better design decision-making, rationale and reuse. For instance, a novice engineer can discover not only what features were defined in a previous design but also trace back why earlier engineers designed those features in a specific way. It also provides a means

to interrogate the engineers' discussions, interactions and cognitive processes during earlier designs.

This case study also demonstrates how the captured temporal meta-data can be represented in various formats including IDEF and DRed as exemplars to name but a few. Although structuring and organising the captured information into various representations is important, the thinking process of the designer or the design ideation does not follow a specific structure or sequence [186]. As a consequence, the holistic activity capture enabled by the multimodal framework and inherent process mapping provides new opportunities for exploring methods for structuring and organising information which subsequently, after interpretation and understanding, becomes knowledge.

Conventionally monitoring CAD user activity is mainly based on the log files created by the CAD software. The UbiITS framework enabled using contemporary, sophisticated computer-based tools to capture of user activity in temporal manner and provided opportunities for understanding CAD design activities to unprecedented levels of detail. Multimodal ubiquitous data captured through a variety of devices, including CAD software logging, should be merged if meaningful data about the CAD activity is to be better understood.

5.9.1. Ubiquitous metadata capture environment

Different representations of the experimental CAD activity and bio-physical data synchronised with the events occurring in the design environment demonstrate that capturing the design activity using ubiquitous multimodal media can provide a better insight into the actual design process. The ubiquitous system presented in this study enables capturing multimodal metadata, unobtrusively, i.e. without inhibiting the process by easily embedding various capture tools in the environment.

Additional concerns were also addressed by the ubiquitous system when deploying in a realistic, unconstrained environment, e.g. a design office with many engineers. Since the design process is virtually a cyclical and long process, the data capture also needs to run continuously over a long period of time. The distinctive modular-based design of the framework enables it to run indefinitely. A base station or a centralised control panel is required to run uninterruptedly while individual device units can be started and stopped as required. Since the framework allows the plugging or removal of devices at runtime and caters for device failure, this makes the framework robust and suitable for a rugged environment such as a design office.

5.9.2. Interaction and Synchronisation

In addition to capturing data, units that interact with the user or the environment can also be linked to the framework; the framework would treat them indistinguishably from capture tools. Given that a unified interface has been provided for an interaction unit and uses the data flow modules to access data from other devices, it blends into the framework seamlessly. This feature potentially provides opportunities for knowledge push during a CAD task. Previously captured activity and the knowledge committed to a knowledge base can be used to offer information to a designer automatically as and when the designer needs it. A real-time system actively monitoring for user activities can identify patterns of activity and retrieve information from similar previous activities captured and formalised in the past.

This study has proven that the framework synchronises multimodal sources and evidence provided to demonstrate the achievable accuracy of synchronisation. In particular, when considering the technical requirements for a prospective brain computer interface (BCI) system for CAD, accurate synchronisation is critical between the operating system, the CAD application, the BCI device (EEG) and other modalities. This is fundamental because a BCI CAD system must be responsive but yet be able to tolerate various device protocols. The evaluation of the CAD system showed that the proposed framework is capable of handling this requirement through capturing and mapping EEG for design activities. The present study uses medical grade EEG and other psychophysiological sensors for safety, data provenance and hygiene reasons. As we understand more about design activities using a virtual environment (i.e. CAD) so too will the development of collaborative design, BCI controls and indeed how the functionalities in the virtual environments or CAD systems can be improved. Therefore a ubiquitous framework such as this addressing the fundamental technological challenges becomes very useful.

5.9.3. Handling vast quantities of metadata

Appropriate real-time fusion and processing modules can be incorporated in the system to produce ready-to-use data; resulting in a reduction of stored information compared to the raw data. For instance, a module for a video camera capturing the user activity only needs to store the recording when there is a presence of the user. Therefore plugging such a dynamic component into the system can possibly adapt the tools capturing that data, e.g. increasing or decreasing the data resolution. Early filtering of the captured content prevents inappropriate information being accumulated into the repository.

Currently the metadata and events are stored locally in individual files. Although this technique caters for the high bandwidth and low latencies, it causes difficulties when the

captured data is accessed in real-time or shared with multiple users. One solution is to combine storing the captured data in centralised repository. Balance between distributed local files and centralised repositories will thus be needed to take the advantage of both techniques. Furthermore, organisational structures might require control over what is committed into the repository and moderation steps on the captured metadata before it is committed into mainstream repositories.

5.9.4. Integrating metadata capture in existing and future CAD solutions

Present-day CAD solutions are typically not equipped for multimodal metadata inputs. A metadata capture system cannot be limited to ad-hoc configurations; instead it needs to be limitless with regard to the number and types capture devices. An ideal unbounded system should be open to interfacing with various multimodal tools and support different configurations of the metadata capture system. Therefore a future CAD system with metadata capture must adopt standard protocols such as the ubiquitous framework proposed in this thesis. This will permit future CAD systems not to be selective on tools but to accommodate diverse multimodal tools to capture activities and interact with the design process. Consequently the users may select their own capture tools and mount those onto the system to construct a customised configuration of a metadata capture system.

Although PLM systems provide few facilities to handle multimodal data, e.g. audio, video, hyperlinks to other data types, etc., they are obtrusive, as a user needs to upload the information collected during the design. This interrupts the flow of the design process and somebody has to do the additional work of capturing and gathering the metadata. In contrast, the ubiquitous framework outfits the design environment with embedded data capture tools and hence provides a metadata collection system which is unobtrusive to the process. Two possible configurations are proposed for using the framework for metadata capture in a design environment.

1. Embedding the metadata capture tools within the existing CAD systems

The framework can be used to extend the existing CAD solutions to include ubiquitous metadata capture, whereas multimodal inputs will be captured alongside the CAD design trail (Fig.5.14). This is similar to the case of metadata capture using a CAD station as demonstrated previously.

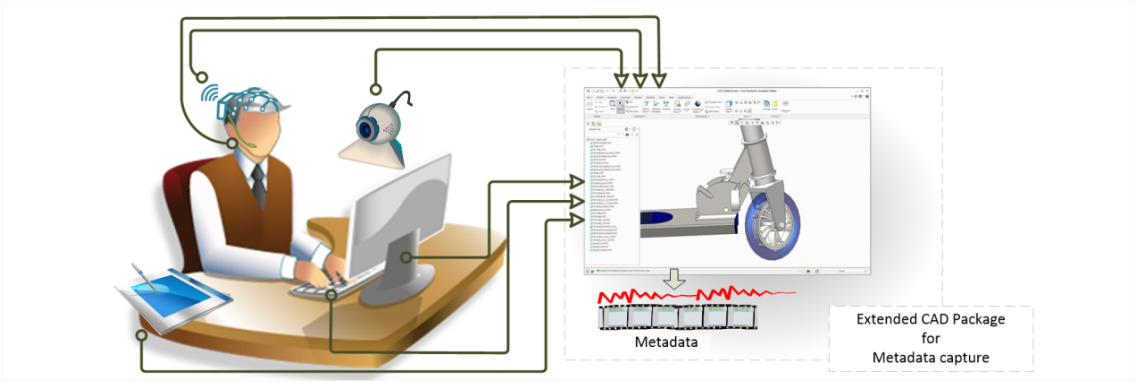


Fig.5.14 Extending CAD station for multimodal metadata capture. The CAD package can be customised using the framework to accommodate multimodal tools.

2. Building a metadata capture infrastructure based on the ubiquitous framework.

A complete metadata capture infrastructure can be constructed using the framework. This will comprise ubiquitous data capture tools and multiple CAD stations. In this arrangement any CAD stations will be considered as an element of the whole system.

In both cases the metadata and updating PLM relations is driven automatically by the framework. Embedding the metadata capture tools within standard CAD systems is a quicker and a simpler way to implement a metadata system but results in limitations over the extendibility. This is more suited for a single user CAD station (single user) but is limited to the customisation features such as an SDK or API provided by a proprietary CAD solution, e.g. PTC Creo TOOLKIT, Siemens NX/UG Open API. In contrast, constructing a complete metadata infrastructure is more suited for multi user, parallel sessions and distributed sites metadata collection system. Thus, it comes with the cost of implementing a complete ubiquitous capture infrastructure and adopting the information generated into the current PLM systems, i.e. raw metadata such as audio files, video files, and key press logging or processed data such as filtered activities and emotional states.

5.9.5. Feeling of being watched

Capturing metadata during various stages of the design activity provides unprecedented insights about the design process and traceability of the design solution. However capturing user activities can potentially have adverse effects on the employees (i.e. designers, engineers) such as being suspicious about the employer and possibly causing a breakdown of trust. This topic has been discussed in several studies with a view to addressing the shift in technologies, workplace privacy and ethics [187]–[190]. The framework allows capture devices to be enabled and disabled by the user in runtime, i.e. being dynamic and reconfigurable as opposed to being a fixed system. This makes it possible for an engineer to walk into the ubiquitous environment

and start (or stop) capturing his/her own work. Therefore it imitates an atmosphere where an individual can sign-in and sign-out allowing individuals to have control over the capture of his/her own activities. It is also possible that the captured metadata might be used to evaluate the employer performance. However, this issue is associated with the organisational and social aspects of how this information is used, whether for revisiting and tracing of the design knowledge and product information or for evaluating employee performance.

5.10. Conclusion

This case study has presented novel ways of capturing metadata within a design process beyond the standard design records such as CAD parts, geometries, and manually produced reports and minutes. The metadata is captured and stored using ubiquitous multimodal capture tools embedded in the design environment. This data capture is automatic and blended into the habitual design activity and therefore unobtrusive to the process. Consequently, the designer/engineer experiences no interruption in the usual design activity and most importantly, there is no extra workload involved in capturing and generating the meta-information about the design process.

The UbiITS framework establishes the metadata capture system and drives the automatic and ubiquitous capture of temporal multimodal data. The framework forms a highly expandable system with systematic arrangement of data capture tools. It provides a standardized, repeatable method for integrating multimodal, ubiquitous devices and hence enabling the progressive construction of a complicated data capture infrastructure. The approach maintains a generic architecture for multimodal data capture and enables construction of systems for metadata in diverse environments with disparate obligations. In contrast to related studies, e.g. DAKA [159] and KEN [157] which deal with the captured metadata, this work provided a detailed technical resolution for capturing multimodal metadata and demonstrates an end-to-end pipeline from the capture and transformation of raw measurement data into metadata representations mapped onto the design process, all accessible and generated automatically.

Chapter 6. A Virtual Aided Design Engineering Review (VADER) system with searchable content: Knowledge engineering via real-time multimodal recording

Related Publications

Aparajithan Sivanathan, James Ritchie, Theodore Lim, and Raymond Sung. "A Virtual Aided Design Engineering Review (VADER) System with Searchable Content: Knowledge Engineering via Real-time Multimodal Recording" (n.d.). – *Submitted Article to Journal of Computing and Information Science in Engineering*

S R. C. W. Sung, J. M. Ritchie, T. Lim, A. Sivanathan, and M. J. Chantler, "The evaluation of a Virtual-Aided Design Engineering Review (VADER) system for automated knowledge capture and reuse," in *Proceedings of the ASME 2013 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference*, Portland, Oregon, USA, 2013.

J. Ritchie, T. Lim, R. Sung, A. Sivanathan, C. Fletcher, Germanico Gonzalez, Hugo Medellin, Ying Liu, and Zoe Kosmadoudi, "VES: Knowledge Capture in Virtual Reality and Beyond," in *Advances in Computers And Information In Engineering Research*, ASME, 2014.

Recent research has found that knowledge acquisition is frequently not performed in engineering companies due to time and cost restraints, even though industry recognises and acknowledges the benefits [191]. As a consequence, knowledge databases in their product lifecycle management (PLM) systems tend not to work to their full potential [192], [193].

Capturing engineering knowledge and making it immediately accessible has been a long-standing challenge and widely-researched area [193]–[197]. The focus on automating the process in order to reduce the associated time and costs involved was seen as a means to promote a greater adoption of these methods in industry [198]–[200]. Yet the main obstacles to date still involve filling PLM systems with data that is manually created by hand e.g. handwritten notes, logbooks etc. or data files e.g. CAD files, documents, plans, etc. which are poorly linked and leave no context or mapping. This is time consuming and also often leads to knowledge that is too focussed on a very particular area [201]. Additionally, inadequate knowledge capture can reduce the amount of knowledge that is reused in a company [202] leading to time consuming searching or costly "reinventing the wheel".

A promising approach utilising virtual reality to capture and formalise knowledge at various stages of the design and manufacturing process has been demonstrated in previous work by Ritchie et al [203] and Robinson et al [204]. Subsequent work by Sung et al [205] demonstrated

how detailed logging and analysis of design actions could be used to analyse design activities, and further map the associated knowledge articulation onto the process.

The Virtual Aided Design Engineering Review (VADER) system reported in this chapter follows this trend of utilising virtual reality (VR) with the focus on multimodal tools to capture knowledge during the design stage; the emphasis being on engineering design review meetings, an area often overlooked in knowledge engineering research. These types of meetings represent a large proportion of the design stage where a significant amount of valuable and important knowledge is generated that can potentially be reused in future projects. To illustrate the cost impact due to a lack of knowledge reuse, a US Navy-commissioned report on shipbuilding revealed that the overall cost of two projects increased by 45% and 83% [206], caused by the poor knowledge reuse in the design stage and led to redesigns having to be made.

6.1. Current state of knowledge capture in industries

Description	Number of Companies out of 7
Practising any knowledge capture	6/7
Know the importance of knowledge capture?	7/7
Need to revisit previous designs?	7/7
Document design decisions during/after design session?	6/7
Best practice guide?	7/7
Difficult to find knowledge?	5/7
Problem with lost knowledge when engineer leaves?	4/7
Object to being logged during design sessions?	2/7
Method of sharing knowledge in company	
Intranet portal/ Wiki	2/7
PLM Systems	1/7
Minutes	3/7
Informal Discussions	6/7
No Obvious knowledge sharing options	1/7
Opinion of automated knowledge capture - Usefulness	7/7
Time spent on capturing design knowledge known?	0/7
Money spent on capturing design knowledge known?	0/7

Table 6.1 Summary of industrial visits showing issues related to knowledge capture. This tabulation includes extended information to an earlier study by authors [169].

In order to gain a better understanding of the current state of knowledge capture and design review methods in the UK engineering sector, 7 major UK international engineering companies with worldwide sites were contacted as part of a case study. The study included site visits and observations during 'live' design reviews followed by discussions on the knowledge acquisition methods used. Table 6.1 summarises common knowledge capture issues present in the

companies. Note that this includes additional information which extended a previous study outlined by Sung et al [169].

The following points summarise the key findings:

- Manual or semi-automated design review methods were in use by the 3 companies;
- Virtual reality was used during design reviews by only 1 company;
- Design reviews were considered to be time-consuming and not comprehensive by the 3 companies;
- The knowledge search facilities used by 2 companies were deemed to be inadequate;
- Engineers in all the companies were using logbooks to capture data, but the uploading of the data (typed in or scanned) to the company database or PLM system was not always enforced;
- One of the companies mentioned that speech recognition and the electronic mark-up of drawings would be useful additions to their current design review process;
- Where companies did not comment on particular knowledge engineering issues, then this was further questioned and clarified as there were no obvious methods being used in practice to mention;

To ascertain the potential cost savings that could be achieved with a more efficient design review process, it was estimated by one of the visited companies – a multi-national aerospace company – that they hold 240 major design review meetings annually, and each of these meetings costs approximately £27,000 in time and support effort, i.e. 240 x 27000 per annum. Consequently, any means that can be found to improve the efficiency of these meetings could potentially lead to considerable savings.

Even though commercial design review systems are being used, all industrial collaborators have noted several important features are missing from them. For example, Nvivo [207] can capture audio and video data but does not support CAD file formats. Alcove 9 [208] and AVEVA [209] can both annotate and visualise CAD models but do not allow capturing of audio or video. This has led to continuing research into ways to improve current design review practices. One example involves a system using Microsoft Kinect trackers and touch-enabled devices to provide interaction [210]. However, this work does not focus on how the data capture and analysis is performed. Augmented reality to enhance the design review process in the engineering and construction industry has been investigated where design reviews can be

held with co-located and remote users [211]. The findings revealed that while task completion times and mental workload were reduced use of the system led to increased physical discomfort for the participants. The IAP Design Review also uses augmented reality to record audio, video and annotations and physical prototyping could also be carried out [212]. Drawbacks of the system include the use of bulky hand-held devices that users are required to use, and annotations cannot be viewed during the replay of the captured audio and video.

Knowledge Enhanced Notes (KEN) system by Conway et al [157] was developed to enhance the documentation of activities in engineering meetings. This system supports predefined meeting structure templates and file drag-and-drop uploads to record document exploration activities take place during the meetings. A multimedia timeline representing the document access was included in this system that can be used to trace back and follow the information flow during the meeting. This work substantiates the importance of temporal multimodal data for revisiting and understanding the design meeting.

A major problem with existing design review methods is that they are obtrusive and time consuming, often requiring someone, usually an engineer needs to recall and summarise the knowledge generated during the extensive review discussions and manually entering it in to the knowledge management system after formalisation and agreement with participants. The actual review activity and the knowledge capture/management system are separated and reside as two distinct entities in this method. Contemporary systems do not integrate the time-phased synchronous capture with the real-time activity. Consequently these systems tend to function simply as data storage or document management systems. In contrast, VADER system presented in this thesis integrates and automates the knowledge capture and management process into the inherent review activity.

6.2. The Virtual Aided Design Engineering Review (VADER) system

A domain analysis on the existing design review methods and tools revealed limitations that established the fundamental requirements of the VADER system. Many review activities are conducted during the engineering of products from major formal design reviews to formal and informal specialist reviews, including at individual engineer level. VADER is primarily focused on team-based design reviews; however, the concepts and techniques used in the system are generic to other forms of review. Large amounts of knowledge and decisions are articulated during team-based reviews. Capturing, formalising and indexing this collaborative knowledge allows it to be reused later in the project, i.e. via product lifecycle management (PLM) systems and in future projects, therefore potentially providing significant savings both time and cost.

6.3. Process and formal design record linked knowledge capture architecture

The knowledge content exists both in the formal design representations e.g. drawings, CAD files, assemblies [186], [213] and in the live review process [186], [214]. Consequently, knowledge can be captured in relation to formal design representations and the live process activity. Adding annotations to CAD files, collecting and tagging discussions linked to a part, etc. are exemplars of design representation linked knowledge. Secondly, the live knowledge articulation in the design review process can be appropriately linked to the temporal data. For instance, a particular part may be rejected, failed or a compatibility issue is discovered at a later stage of the design because of a modification made earlier; however the earlier decisions that triggered the modification have long since been forgotten and are inaccessible. Chronological data regarding decisions and other information used to make that decision suddenly becomes crucial in this scenario. Capturing and formulating the knowledge with regard to time can be very useful in such a case. In summary, the knowledge tagged in both approaches, namely the formal design record linked and temporal linked approaches have their own benefits and are complementary.

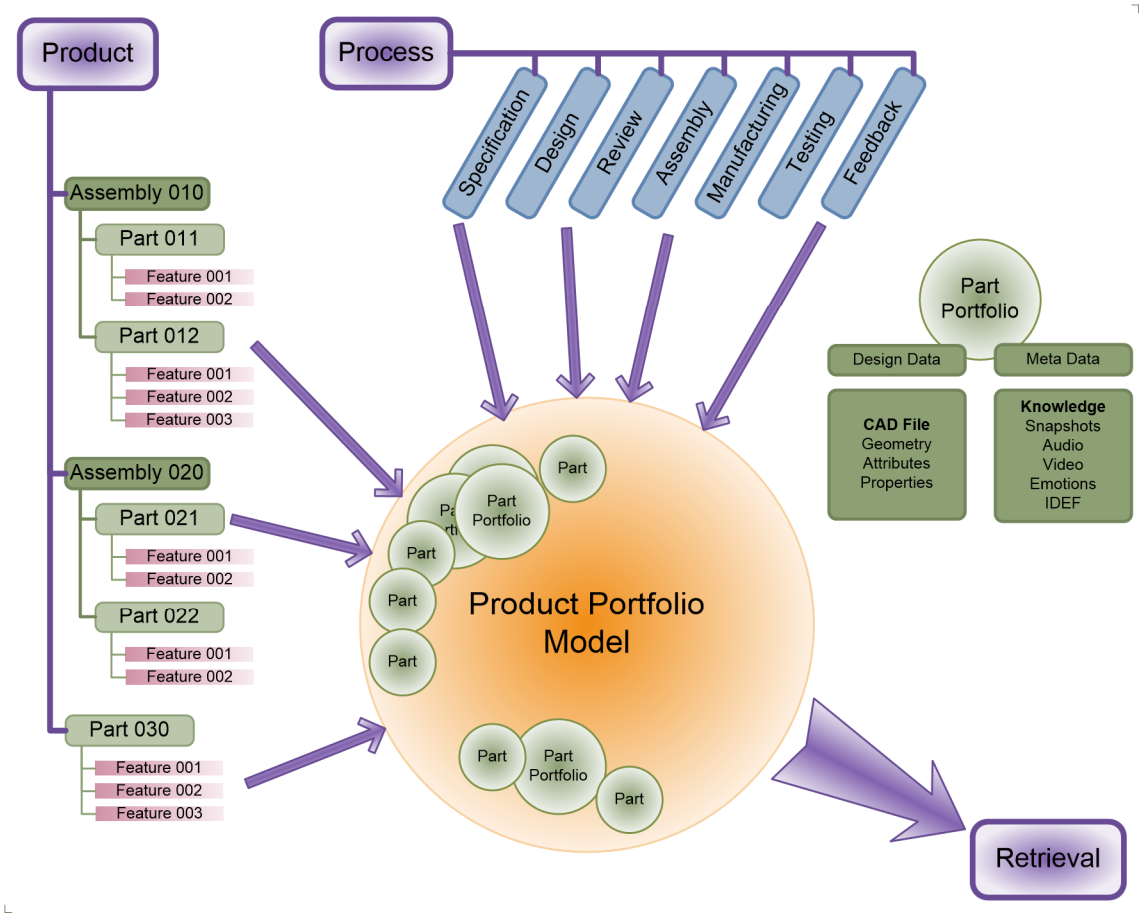


Fig.6.1 Knowledge-capture architecture employed in VADER. This architecture integrates the knowledge encapsulated both in formal design data and the process.

Consequently, as shown in Fig.6.1, the knowledge architecture used in the VADER system employs the knowledge linkages associated to both the formal design records and the temporal data approaches. VADER uses part numbers, PLM record indexes or unique component IDs as the index for the former and the time for the latter. This enables the captured knowledge to be organised both ways and provides bidirectional accessibility. In addition to the textual information, the VADER system captures audio, video and various forms of time-phased interactions during the design review. The interface has been designed for multiuser interaction and therefore supports concurrent interaction. Capturing multimodal interaction data and simultaneous inputs from various actives is inherently a complex task and possess significant technical challenges.

6.4. Application of multimodal data capture and interaction framework - UbiITS.

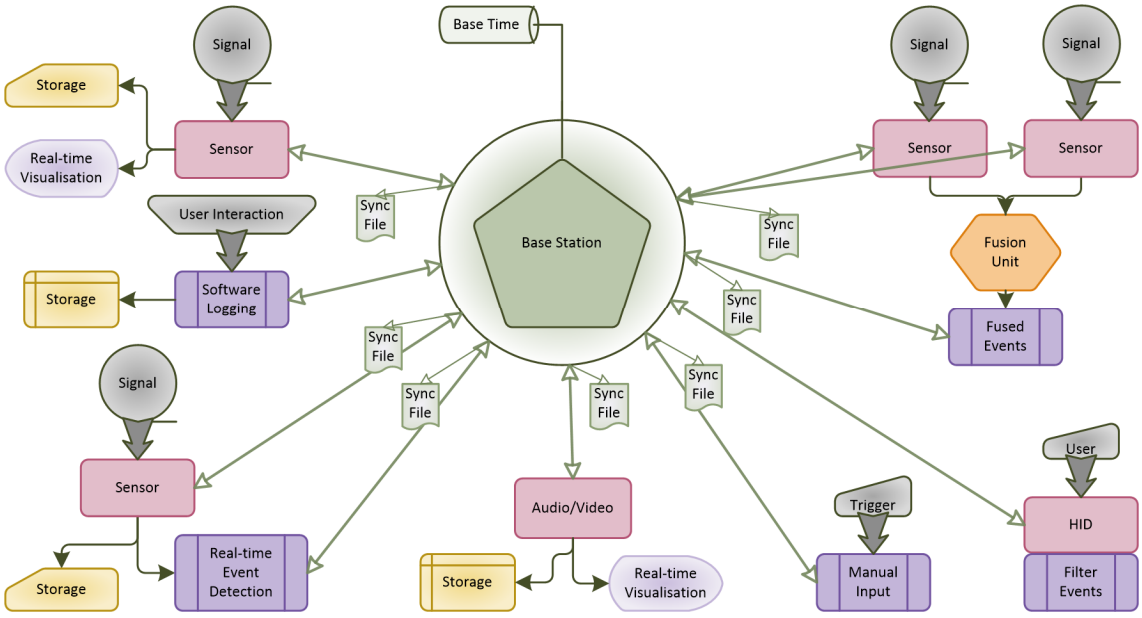


Fig.6.2 UbiITS - addressing the technical requirements related to the integration and synchronisation of multimodal data tools for VADER

UbiITS framework has been employed to address the technical complexities of integrating simultaneous multimodal input and interaction tools in the VADER system. The UbiITS framework predefines the overall structure and design parameters of a data capture system so that if it is large and complex it can be built without having to be distracted by the technical details. This allows the implementers to concentrate on specifics of the system being designed, in this case the VADER system, by reusing the generic modules defined in the framework. These modules embed the guidelines for integrating the multimodal devices, tools and interfaces, providing the systematic structured formation of components into an interactive data capture system. Fig.6.2 illustrates the concept of employing the UbiITS framework,

providing an outline for the integration of various multimodal components and the synchronisation of corresponding data channels.

The advantages of employing UbiITS framework in the VADER-type application are detailed below:

1. **Multimodal Data Interfaces –** Activities are captured via assorted devices and tools. Data delivered from these tools vary in format, frequency, interface, etc. For instance a textual annotation added by a user can be recorded as a discrete action while audio recording is performed by a continuously sampling audio input. The UbiITS framework handles these heterogeneously using predefined uniform interfacing strategies. All data types involved in VADER are classified as discrete or continuous sampled data and handled consistently.

2. **Temporal Synchronisation –** A high precision hardware clock is utilised by the UbiITS framework to acquire 64bit UNIX timestamps with millisecond resolution. Every user interaction in VADER is time stamped by the UbiITS framework. Continuous data channels such as audio and video capture are temporally synchronised based on their frequency. Tight temporal synchronisations are maintained by compensating for temporal latencies involved in individual data channels.

3. **Concurrent and distributed Interaction –** UbiITS enables devices and tools to run on isolated platforms, therefore physical concurrency can be achieved. TCP-based protocols are used by the framework to handle data transport across distributed platforms and additionally to control devices from a centralised platform.

4. **Dynamicity –** The UbiITS framework enables devices to attach and detach at runtime, permitting the VADER system to change dynamically. This feature embedded in the framework is particularly useful since it means that the operation of the VADER system is not disturbed when modifications at runtime are required. For instance, if a user decides to join/leave the system or start/stop audio recording in the middle of the session, the rest of the system can continue to operate uninterrupted.

5. **Extendibility and Interoperability –** Devices and data interfaces are treated by the UbiITS framework in a modular manner. This enables effortless adding of unlimited number of data capture and interaction tools into the system. Any future devices and new data modalities can be integrated into the system with an assurance of interoperability as long as they are programmed in compliance with the interfaces and standards defined by the framework.

6.5. System schematic

The VADER application has been written using C/C++, with the main application including the following major components:

- **UbiITS framework –** for modular interoperation and time synchronisation of multimodal data capture tools;
- **QT framework** [215] **–** for window management, control widgets and network connectivity;
- **VTK framework** [216] **–** for CAD model view rendering and interaction;

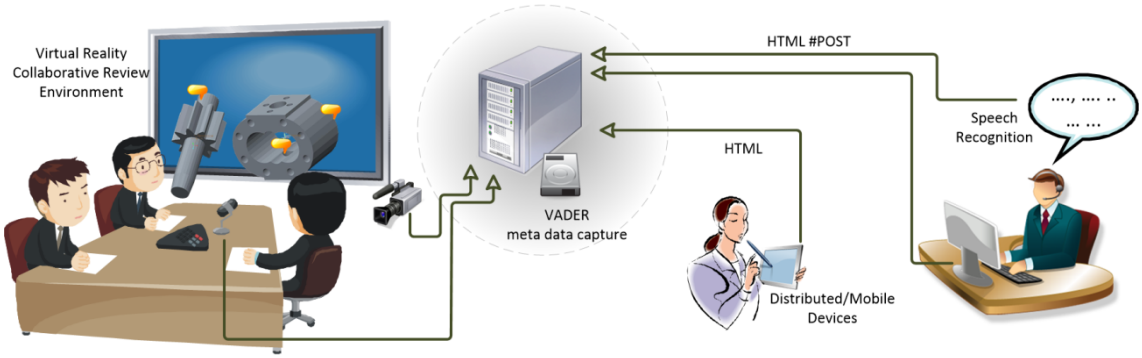


Fig.6.3 VADER system with primary collaborative 3D interface and auxiliary web interface

6.6. CAD model view and interactions

The VADER system primarily consists of a virtual reality 3D interface based on a Full-HD (1920x1080) 3D projector and 3.2m x 1.8m power wall projection setup. Active shutter glasses enable 3D rendition of the CAD models and assemblies (Fig.6.4). The large power wall display provides a greater immersive experience and a more conducive environment for collaborative discussions. Alternatively, this interface can be switched to desktop or 2D modes, e.g. desktop 3D screens standard 2D displays for portability and affordability.

Fig.6.4 VADER System in use – Industrial Partners using the VADER system to review their own designs

CAD files and assemblies can be loaded into the VADER system as pure geometry files. To maintain compatibility across various CAD packages, proprietary CAD files are pre-converted

into STereoLithography (STL) files. While the standard geometry data is loaded through STL files, auxiliary properties such as colour, transparency and lighting effects can be set and loaded through XML configurations. Specifically, XML files are used to implement the hierarchical relationships among assembly, subassembly and part files. Parent-child nodes in XML enforce the hierarchical assembly relationship tree. Position and orientation information of subassemblies and parts are also integrated in the XML files to ensure all components are loaded in the appropriate locations.

Once the assemblies or parts are loaded, users can view and interact with the models as per standard CAD applications, e.g. pan, zoom, rotate, select standard/predefined view orientations, etc. Furthermore, users are able to perform entity or assembly related operations such as selecting and highlighting components, hiding/showing components, by directly interacting with the visualised model or using the part/assembly tree view (Fig.6.5).

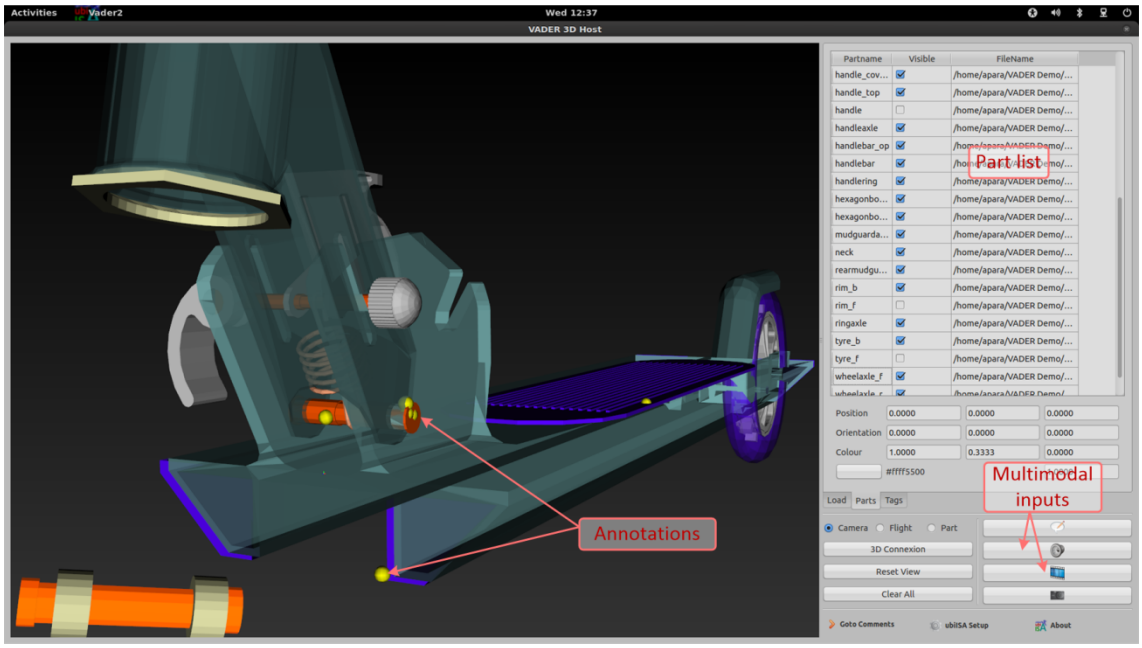


Fig.6.5 VADER's virtual reality based 3D CAD model view interface for collaborative interaction and runtime annotation.

6.7. Review interface and multimodal interactions

VADER system through the use of UbiITS allows connecting up multiple keyboards and mice so that multiple participants in the design review environment can interact with the model using their individual keyboard/mouse. This feature allows every participant to interact with the model and hence significantly enhancing the collaborative and team nature of the review process. Alternatively, to prevent unsolicited interjections, a moderator may perform actions on behalf of other participants, possibly a chairperson supervise all the actions with another set of controls.

VADER provides the important means of tagging annotations to individual parts, subassemblies or even whole assemblies. A tagging module extended (inherited) from generic UbiITS device modules was used to capture annotations with temporal data. This module is capable of capturing temporal data from distributed input nodes in a synchronised manner. Various user interfaces such as VR-based component tagging, a window-based textual annotation box and a web form were attached as frontends to this module; these are detailed at a later stage. However, the core functionally of temporal tagging remained unchanged and powered by a generic UbiITS module in the backend.

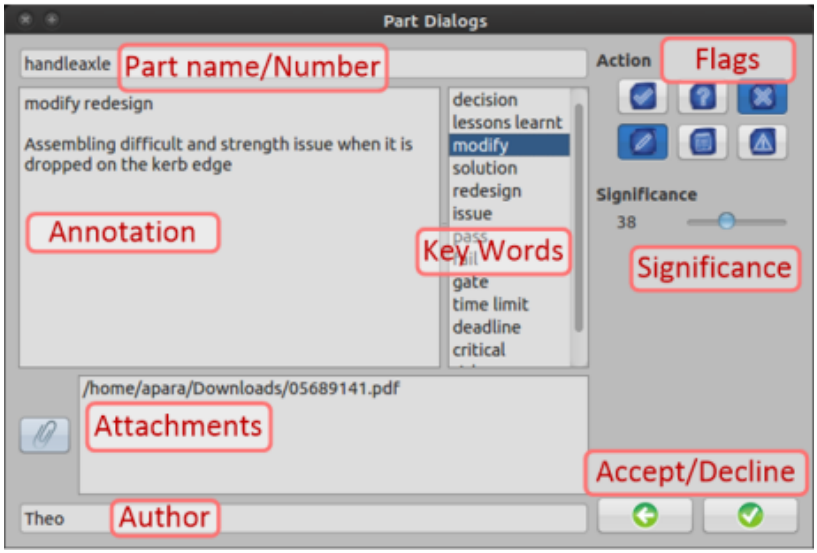


Fig.6.6 The VADER annotation dialog box provided for adding textual annotations also includes fields for follow-up flags, significance value, part number, etc.

Participants interacting with the models can add annotations intuitively by clicking on the components. The annotation text will be tagged automatically against any selected component at the exact 3D coordinates where the participant wanted to add the annotation. An annotation box provides advance options to attach documents, select required follow up actions, quick access pre-defined keywords, denote a significance value (1 -100) and the author's name (Fig.6.6). Additionally, user registration can be used to input the author's name automatically. Predefined keywords are loaded by the system from a configuration file comprising a collection of the most used words e.g. decision, issue, redesign, deadline, lessons learnt etc. or commonly used terminologies in company-specific or product-specific review meetings, e.g. for design stages, passing design to another phase through a gate, etc. This configuration file can be easily amended in accordance to the company or type of review. The keywords are quickly accessed by clicking on them or using an auto-completion feature. Follow-up flags are mainly indicative symbols for future reference and associated to the necessary actions to be taken regarding a particular annotation, e.g. pass, fail, revisit,

information, warning, etc. Significance value is another indication provided in the annotation box to express the reviewer's or teams feeling about the importance of the annotation.

In addition to textual annotations, synchronous audio and video can also be captured selectively or automatically depending on user preference. Audio from a conference microphone and a video camera capturing the review room can be attached to the system for this purpose. As stated in the section 6.4 multiple audio and video streams can be captured simultaneously by the underlying UbiITS framework allowing every participant to have their own microphone and camera, optionally having control over the capture of their own actions. The system also captures the model visualisations as screenshots automatically whenever any interaction with the model is performed.

6.8. Distributed web Interface with speech recognition

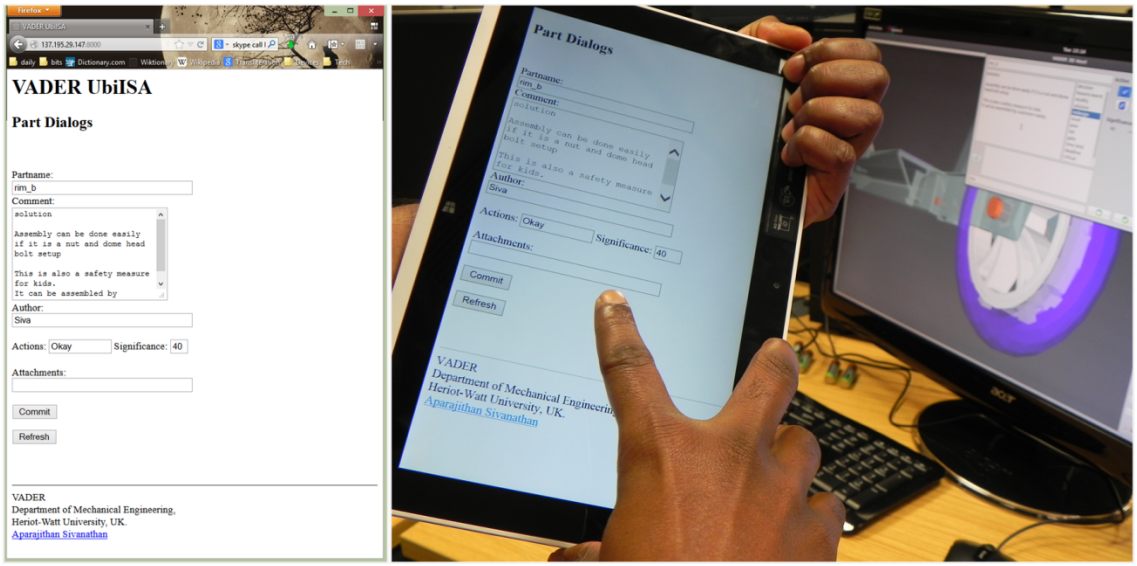


Fig.6.7 VADER Web interface

VADER is also equipped with a web interface (Fig.6.7). The primary VR interface is essentially a single user interface but permits multiuser interaction through separate key boards and mice. Although this interface provides a centralised display to everyone, it is not suitable for multiple users interacting simultaneously, i.e. everyone adding their own annotations simultaneously. The web interface was developed for the purpose of simultaneous and distributed multiuser interaction. Any user can connect to the web interface via their own web browsers, typically using their personal laptops, tablets or phones. The annotations are sent to the VADER system using standard HTTP compliant posts, where the VADER is integrated with an HTTP server constantly listening for connections and HTTP post. These distributed interfaces running on detached platforms naturally use asynchronous clocks. The temporal synchronisation of these

distributed interfaces is handled by the associated generic technique defined by the UbiITS framework (see: 4.6.2).

This feature is ultimately designed for enabling the concept of concurrent engineering, with multi-user multi-site capabilities. Users connecting to the web interface have a web page similar to the annotation dialog box. This allows annotations to be added concurrently at any instance of time as is necessary. The web interface provides one very important advantage in that a user does not need to be physically present in the review room; although, if they so wish, they can use it at the actual review room. Present day speech recognition tools need to be trained per individual and it is difficult to have a system with a multiuser shared speech interface. To circumvent this problem the VADER web interface allows every user their own device and hence with every participant having this, they can also be equipped with a personal exclusively trained speech interface.

6.9. Timeline, data search and retrieval

Captured data is stored by the VADER system in two main formats: XML and binary. XML format is generally used for textual information, such as annotation, follow-up flags, author name, etc. Other multimodal data is stored in a plain binary format since it is significantly faster and compact. Additionally the time synchronisation data, i.e. 64bit UNIX timestamps from the UbiITS framework, is stored into binary format. Basically it is more efficient to store any data that is textual into XML records and those are only machine-readable into binary data.

Since UbiITS captures every interaction and multimodal in a synchronous manner, VADER can exploit this temporal synchronicity of multiple data channels to populate an integrated, global timeline with all-inclusive interactions and multimodal data (Fig.6.8). The timeline is aligned with a real-time clock that is used by the UbiITS framework and includes all interactions performed with the CAD model, audio/video recordings of the discussions, annotations added, screen rendered images of the model view, etc.; ultimately, providing a comprehensive, detailed and accessible record of the design review meeting.

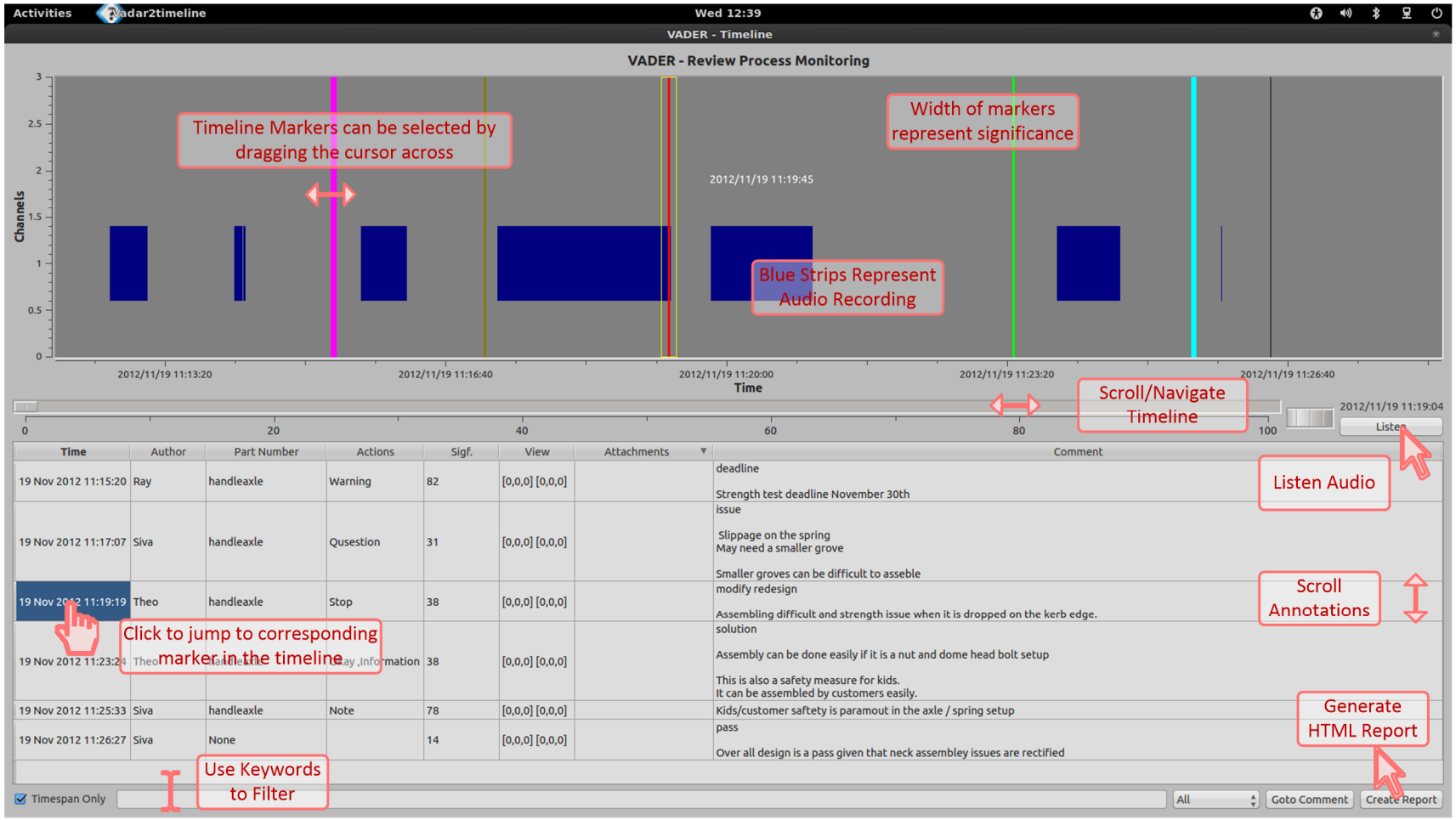


Fig.6.8 A multimodal timeline provided for navigating through the captured data, searching and filtering. HTML reports can be created instantaneously based on entire or filtered data

The timeline and the list of annotations can be accessed after or during the review session. Users can scroll and zoom to view the contents of the timeline. When the user selects an annotation flag in the timeline the associated annotation will be instantaneously highlighted and vice versa. The annotation list can be sorted based on various fields, e.g. time, author, part number, part description, significance, follow-up flag types, etc. Moreover the annotations can be searched using the search box provided, which allows searches based on chosen fields or entire selection; it also includes the wildcard and regular expression search.

6.10. Automatic review report creation

One factor on the low uptake of knowledge capture reported by industrial partners is that compiling information for reports is hindered by the many laborious manual activities and the need for cross checking. The VADER system provides a means to automatically create a report with rich contents at a single click. Anyone searching through the captured data can create an HTML report integrated with multimodal data, e.g. audio, video, screen images. Prospects of creating a report can vary largely, amongst many combinations, these include creating a complete report that includes everything happened during the review session, a report that covers a selected time span, searched and filtered reports specific to a component, sorted reports from high significance, etc. A sample of automatically created HTML report is shown in Fig.6.9.

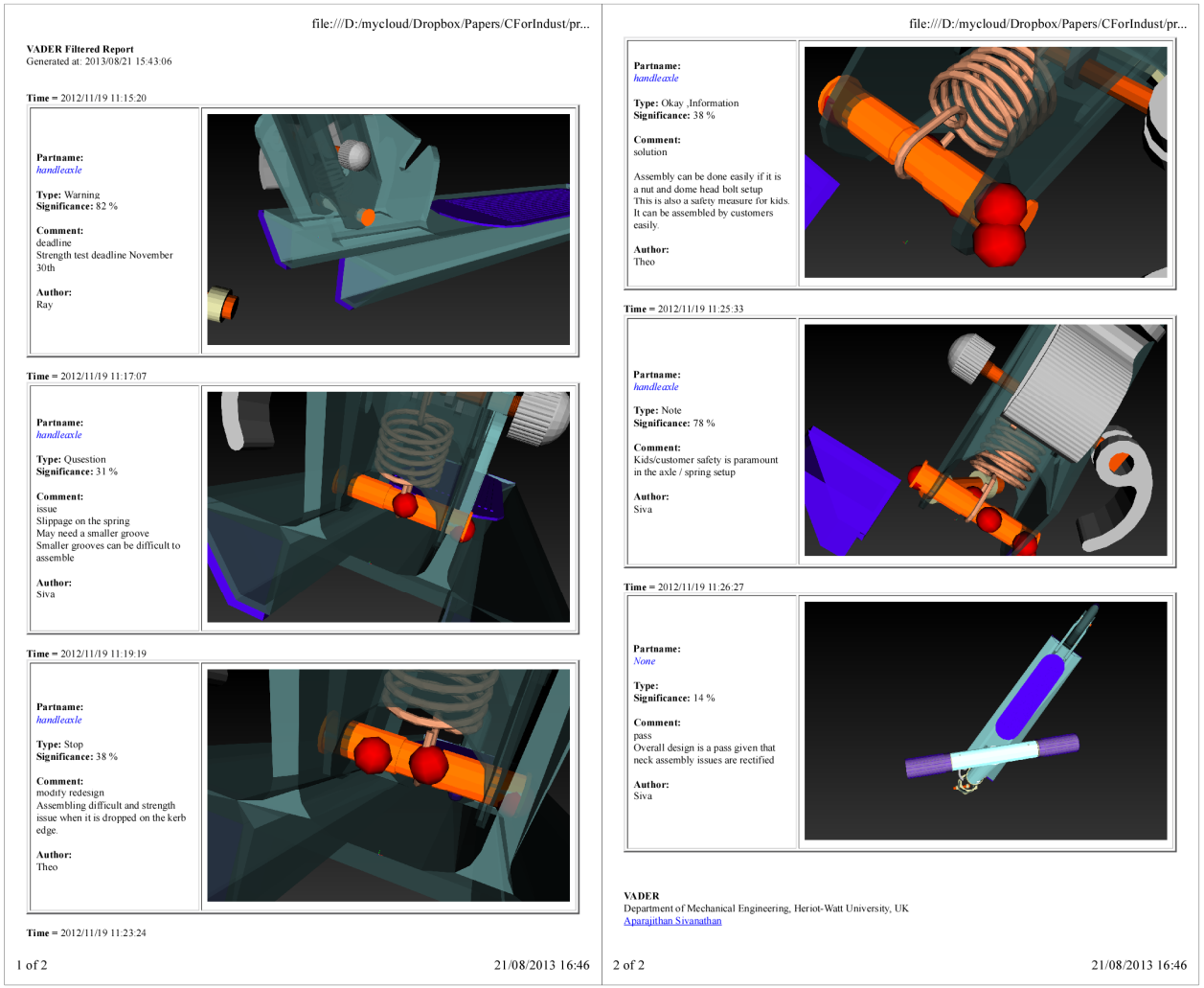


Fig.6.9 Automatically Generated HTML design review report

6.11. Bidirectional linkage to design data and temporal multimodal data

Interactions performed with the CAD models and the multimodal data can be related using their temporal associativity. This in turn allows mapping the multimodal data with a particular component or assembly being accessed. Associating the temporal multimodal meta-data with the CAD models enables fast and accurate knowledge and information retrieval, particularly when linking the temporal multimodal information to the formal CAD design/PLM records. The captured meta-data with the temporal and PLM record associativity provides both bidirectional linkage and accessibility such that the captured information can be traced and reused using both time and formal PLM records. Various techniques to retrieve the information with example scenarios are described in Table 6.2. Furthermore data can be easily integrated with standard PLM systems using the PLM record linkage.

Retrieval Type	Navigation Indices	Example Scenarios
Replay	Time	The captured multimodal data can be played back for the whole review session or for a selected time span
Explore	Annotation text	Navigate through textual annotations and jump to specific moment of time the annotation/interaction was performed and query the multimedia data to obtain in-depth understanding.
Formal PLM Indexing	Part Number / Component ID	Find all the annotations and multimodal data related to a specific part. All discussions and interactions related to a specific part or component can be traced back, read and replayed.
Sort	Significance	Captured records can be sorted based on the significance value given by the reviewer to get an idea about the impact about the particular issue or information
Filter	Follow-up Flags	Extract all the issues that caused the design to fail in the review.
Search	Keywords	Search for a particular keyword or set of keywords used during a review sessions. E.g. customer complain, safety, wear, corrosion, etc.
Filter	Attachment	Search for all the discussions related to a particular external attachment, e.g. emails, reports, documents, images, etc.
Multimedia Search	Multimodal data	Audio, video data can be analysed using cutting edge algorithms for finding historical trends and modelling the contents e.g. finding the words those have been mostly said during a period

Table 6.2 Various methods to retrieve the data become possible with the structured and temporally synchronised data captured by the VADER system. Example scenarios of data retrieval are given to demonstrate the advantages of manipulating the multidimensional indexing and navigation.

6.12. Node - link organisation for design and meta data

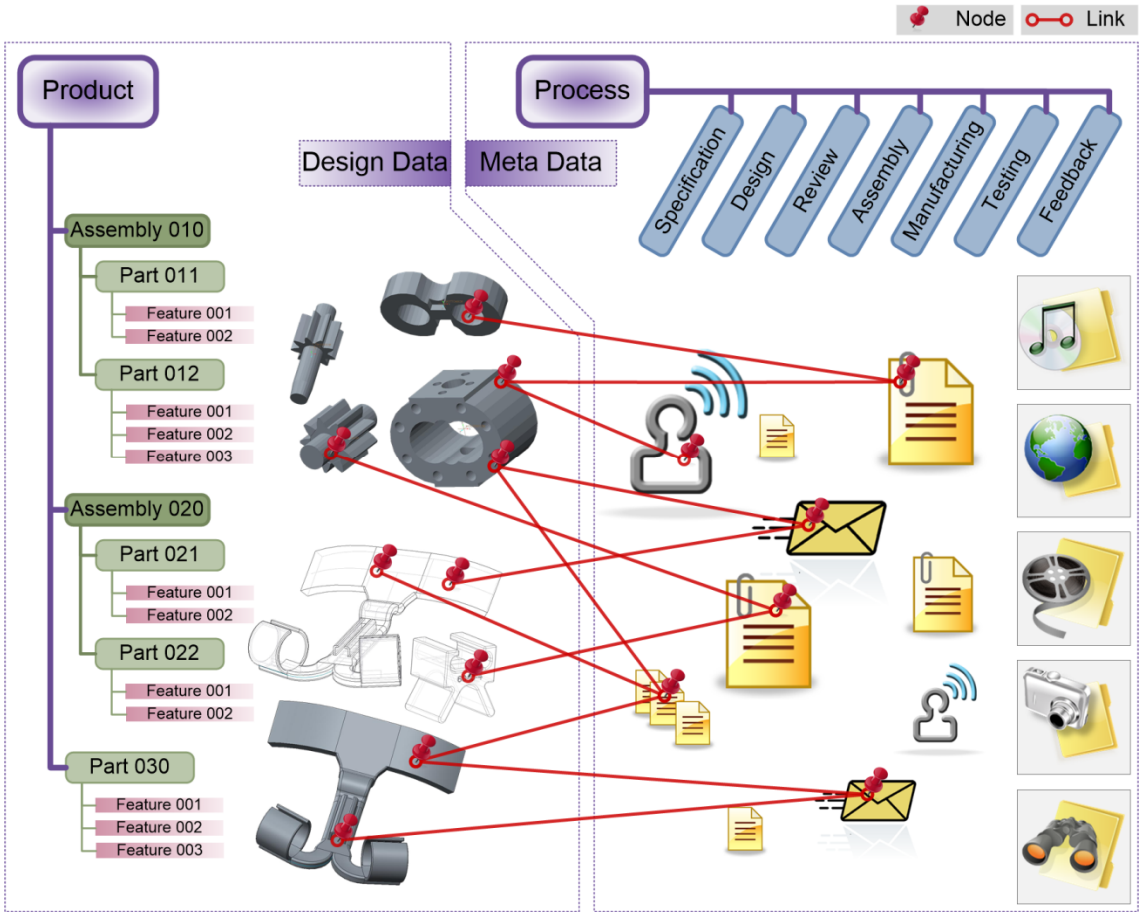


Fig.6.10 Node-Link based data organisation for representing the associativity between formal design records and the temporal multimodal meta-data.

Fig.6.10 illustrates the nodal organisation in the VADER system to maintain the associativity between formal design records and the captured meta-data. Every component in a design (assembly, part or feature) and the captured multimodal data (audio, video, annotations, emails, etc.) are denoted as nodes. An interaction performed during the design process associated with a design record is represented by a link, where those interactions are always be linked to the actual time they have been performed. Therefore the VADER system records the time of interactions as key information along with the associated design data and meta-data. These links are later used as a network of interconnections for the purpose of navigation and information retrieval. Beyond preserving the interconnection between design and meta-data they can be reinforced over the product life cycle based on the post capture usage pattern. For instance if a property of 'strength' of links is introduced in the data management, it can be correlated to the importance based on how many times it has been accessed to retrieve some information and incremented based on every time it has been used to navigate. Search algorithms can be designed to give more importance to stronger links and consequently produce results inclined to the stronger links, therefore delivering efficient search and

navigation experience. In a simpler approach, the significance factor in the annotations (Section 6.7) is correlated to the strength of a link to demonstrate the concept.

The captured meta-data is initially raw in format and needs to be converted into information and subsequently knowledge by means of various processing, structuring and organising steps. Although structuring and organising the data into various knowledge representations is useful for retrieval, it reasonable to assume that the same knowledge-content will have various viewpoints from different users [186]. Therefore the meta-data retaining the bidirectional linkage with formal design data and temporal multimodal data and the node-link organisation helps to build a comprehensible knowledge store which can serve as a base for subsequent knowledge representations as necessary.

6.13. Usability and functionality evaluation

Both lab-based and industrial user trials were carried out to test the VADER application's usability and the fundamental functionality provided by the system. The user trials involved 10 industrial engineers from 4 companies with each session having the following 4 stages:

1. Using a simple assembly model, the user interface and application features of VADER were demonstrated. The fundamental architecture working in background was also explained.

2. Participants spent 30 minutes to familiarise themselves with the user interface and functionalities. Assistance was provided to get them started with the system and all questions asked were clarified.

3. A scooter model together with a previously-captured design review meeting was viewed by the participants. The task was to revisit the data and understand the main points of discussion in the retrospective review. To retrieve the knowledge from previous review capture participants had to interact with VADER tools and supporting interfaces on model visualisation, timeline, annotations and multimodal recordings.

4. The participants were asked to carry out a mock review using an assembly model provided by their own company. The users were already familiar with these CAD models from their own work, but used the VADER system to review it again.

5. A questionnaire to obtain feedback on VADER's usability and functionality was filled out by participants.

An external video camera was used to capture both the participant and the on-screen activities during the mock review, while a microphone was used to capture all verbal discussions and annotations.

The results of usability and functionality evaluation are shown in.Fig.6.11 The evaluation of the functionality relates to the core architecture and concepts in the system while the usability evaluation reflects the participants' views on the current application interface. A System Usability Scale (SUS) [217], [218] was used to compile the questionnaire results to gauge the usability of the VADER system. An average score of 63 out of 100 was obtained from the 10 experienced engineers who participated, but a particularly low score of 37.5 from one participant caused the average score to be lowered.

The overall feedback from the participants was that VADER system was easy to use and quick to learn. Some comments from the industrial engineers obtained from the anonymously-completed questionnaires and individual interviews include the following.

"Excellent first step and very relevant to our needs. Interested in when it can be used in a real review."

"A step change from previous iterations of interactive review software. Great potential!"

"I can clearly see how this will save my time. I currently lock myself up 2 days a week for organising minutes and preparing documents."

"I am looking forward to see this into our real process."

Participants also expressed their desire to consider more natural ways to annotate and interact with virtual CAD models, e.g. gesture interfaces, 3D pointers, etc. The potential of this system being used as a monitoring tool by the company was also brought up by one participant. However a large number of participants were aware of its ability to track process, legacy and intellectual property provenance. All the users were able to experience accessing the information output from the system, give it context and meaning after interpretation and identify it after reflection as usable context veiled knowledge. Thus supporting both Dalkir's [219] and Davenport and Prusak's [220] definitions of knowledge.

Usability

Question	Strongly Disagree				Strongly Agree	Average Response 1 ⇦ ⇨ 5
1. I think that I would like to use the VADER system frequently	0	0	5	4	1	3.60
2. I found the VADER system unnecessarily complex	2	6	2	0	0	2.00
3. I thought the VADER system was easy to use	0	1	4	4	1	3.50
4. I think that I would need the support of a technical person to be able to use the VADER system	2	3	2	3	0	2.60
5. I found the various functions in the VADER system were well integrated	0	3	3	3	1	3.20
6. I thought there was too much inconsistency in the VADER system	0	4	5	1	0	2.70
7. I would imagine that most people would learn to use the VADER system very quickly	0	1	3	3	3	3.80
8. I found the VADER system very cumbersome to use	1	6	1	2	0	2.40
9. I felt very confident using the VADER system	0	2	6	2	0	3.00
10. I needed to learn a lot of things before I could get going with the VADER system	2	6	0	2	0	2.20

Functionality

Question	Strongly Disagree				Strongly Agree	Average Response 1 ⇦ ⇨ 5
1. I think the capturing of audio is helpful to the design review process	0	0	2	2	6	4.40
2. I think the capturing of the screen is helpful to the design process	0	0	1	5	4	4.30
3. I like the presentation style of the text annotations	0	1	4	4	1	3.50
4. I like the text search engine used to find data that has been captured	0	0	3	6	1	3.80
5. I think representing the captured data in an XML format would be useful	0	1	3	3	2	3.67
6. The captured VADER data will be easy to into an existing PLM or data management system	0	2	4	2	0	3.00
7. VADER successfully captured the whole review process	0	0	3	6	1	3.80

Number of participants	Low	High

Fig.6.11 Usability and Functionality evaluation was performed in VADER system based on feedback of participants from industry. - Number of users and their choices corresponding to the questions asked in the feedback questionnaire are illustrated. Additionally, SUS analysis [217], [218] was performed on the usability feedback and resulted an average score of 63 out of 100.

6.14. Discussion – reflections and opportunities

The VADER system has clearly demonstrated a unique end-to-end pipeline that captures a collaborative review activity comprehensively by means of multimodal data and stores the captured data in a structured format that is automatically accessible. Multimodal data captured in time synchronised manner by the UbiITS helps to draw out a comprehensive picture compared to the information from a single modality, e.g. textual notes. The knowledge captured by means of the multimodal data alongside the interactions with the CAD model helps to deliver an in-depth understanding of what happened and discussed during a design review session. Interactions with the CAD models, discussions, annotations added, etc. all influence and contribute to decisions made at a point in time during the design review. VADER captures all of these activities while they naturally happen with the temporal information, so that the temporal linkage can help to reveal much more about the information than simply a recording.

The review interactions and multimodal meta-data have been captured and stored while retaining links to formal design records. The system demonstrated data capture in a collaborative team based design review session. A key to VADER's multimodal success lies in the interoperable nature of its components. This is the result of the design based on standardising its internal protocols. It should be noted that the concept of VADER is not limited to certain type of reviews but is applicable for other types of review requirements and/or to the entire product engineering and management process. The main innovation of this work is centred on the system architecture that is an integration of the following:

1. Knowledge-capture architecture linking design data and the capture of process meta-data.

2. Using the UbiITS framework to capture the multimodal meta-data with temporal synchronisation.

3. Node-Link based data organisation that associates the design data and meta-data platform.

6.14.1. Architecture extendibility – From design review to product life cycle

Engineering design review process and techniques vary widely across domains, e.g. the model view interface used for reviews in an electronics circuit board-based product or a building/construction project needs to be different to that used for a mechanical design-based product. The concept of reviewing the CAD models will be pointless at an early stage of the design, i.e. the conceptual design stage, where the CAD models do not exist or experience

highly frequent iterations. In these circumstances the PLM data entry indices may also differ, i.e. instead of using 'part number' as the PLM entry index, it needs to be named as something else and more abstract or as a dummy part number. Techniques for managing formal design data also vary across disciplines, e.g. assemblies, sub-assemblies, parts, etc. – for mechanical design, boards, daughter boards, sub-circuits, components, etc. for electronic design and building information model (BIM) records are used in architecture/construction based engineering.

Consequently the core system architecture is the key while the front end application and interfaces are effectively to demonstrate the operation of the core architecture and the concept. The core architecture is usable as a meta-data infrastructure even outside the scope of design review phase, including design, manufacturing and other stages of product lifecycle. For example the annotation and meta-data capture portal can be embedded into the CAD software which is often used in design phases. Similar to multiple engineers reviewing the process, the individual CAD engineer can supply annotations or other inputs for the design, e.g. self-reasoning, justification, etc. This also applies to the manufacturing and maintenance phases where the knowledge relations to manufacturing issues and customer feedback can also be captured as meta-data and integrated into the PLM records. Fig.6.12 illustrates a vision of a complete PLM integrated infrastructure that can be built to capture meta-data over all phases of the product lifecycle. This vision only made possible through the implementation of the UbiITS framework.

6.14.2. Flexibility over multimodal capture – provided by UbiITS

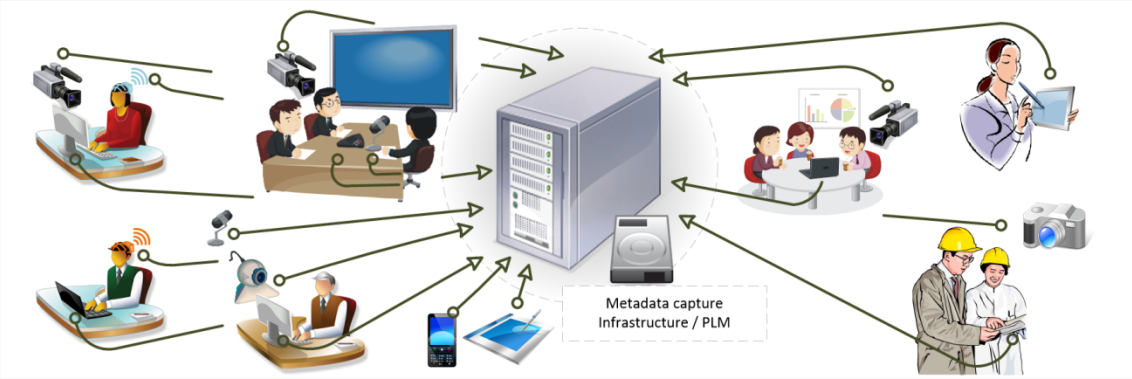


Fig.6.12 Complete metadata capture infrastructure with ubiquitous devices placed in the design environment and also in remote sites

Capturing multimodal meta-data in various circumstances, i.e. in other modes of design review or other phases in product lifecycle, possibly requires miscellaneous types of data. For example the VADER system can be extended to capture data using cutting edge technologies, e.g. free-style hand and gesture interactions, eye tracking, brain control interfaces, etc. The UbiITS

framework plays a vital role here, separating the complexities of integrating these heterogeneous capture devices and data. This allows the core system architecture to operate independently over the assorted data modalities, by handling the multimodal tools uniformly and modular manner; therefore open to any data type and capture devices. The framework also provides the ability to run a time synchronised multimodal data capture system for indefinite period of time, making it usable to build a meta-data capture system running for the entire product lifecycle (Fig.6.12).

6.14.3. Data usage policies and anonymity

A concern was raised by the users regarding capturing activities as the meta-data can be perceived by the users, i.e. designers, engineers, reviewers, as privacy intrusion; particularly those engineers who use the VADER system wanting to have “off the record” discussions. VADER provides features to start and stop audio/video recordings. The UbiITS framework allows recording devices to be plugged-in and removed during run time without affecting the functionality of the rest of the system. This allows users to have control over the capture of their own activity, and the freedom to decide what will be captured into the meta-data system and archived in the PLM system. Alternatively, using aliases and preferring not to disclose personal information at all are other modest choices available to users those who are in a need to reserve their privacy.

The consequences of shift in technologies relating to workplace privacy and ethics has been discussed in various studies [187]–[190]. Although dynamically controlling the recording or choosing to exclude the personal data may work as quick-fix solutions, the fundamental problem of anonymity is related to the company ethics and data retention policies. The VADER system opens real potential for the multimodal meta-data capture and the traceability of design knowledge in the product lifecycle. However the option to use this information purely for tracing back the design knowledge or evaluating the employee performance is entirely the organisation’s choice.

6.15. Conclusion and future work

Far beyond replacing the cluttered note taking activity during design reviews, VADER automates and systematically approaches the process of capturing complex multimodal data and process with the view towards product lifecycle management and engineering knowledge retrieval and reuse. Traditional design review-assisting software packages function merely as data entry systems. In contrast VADER embeds the data capture into the natural design review activity, hence capturing knowledge unobtrusively. The conventional practice of manual note-taking and organising this information into PLM/Review systems after each session is not only

time consuming, error prone and costly but incapable of capturing the knowledge holistically. VADER revolutionises this process by automating the knowledge capture pipe-line, employing time synchronised multimodal data capture tools to apprehend richer information and further systematically organising the captured information for an effective retrieval for the historical knowledge. The VADER approach considerably enhances a similar method of capturing review meeting activities via synchronous multimodal data research by Conway and Ion [157] but differs significantly in that it automates the process by embedding the data capture into the inherent activity and using effective multidimensional indexing method.

Additionally, VADER demonstrates the ability to index the captured data in association with both the time and formal PLM records. The multidimensional indexing and navigational techniques proposed here provide unprecedented ways to efficiently retrieve the captured historical knowledge. This in turn demonstrates the UbiITS approach's capability to produce multidimensional structured data beyond temporally structured data. The generic nature provided by the underlying UbiITS framework makes the VADER system highly extendable and interoperable and not to be limited to a certain types of multimodal data. Exploiting the modularity provided by UbiITS and as an extension from the VADER system, this case study proposed a global infrastructure to capture the knowledge articulated throughout all phases of the product lifecycle.

While the initial concept of employing UbiITS and its potential have been successfully demonstrated, the current VADER system needs further improvement. The current data storage method used in the VADER system works by managing the files locally. It is necessary to transform this simple file management technique to more advanced database systems, so that large quantities of data from distributed locations can be added and accessed simultaneously. Cloud based infrastructures are more suited for these complex systems with added advantages in integrating geographically disjointed campuses and remote work sites.

Future enhancements to the search and navigation methods for knowledge retrieval include the more extensive use and searching of temporal multimodal data e.g. audio/video search, automatically modelling the knowledge and process content [221]–[223] and the integration of IDEF [166] and DRed [167] diagrams, to name but a few. To conclude, the VADER system underpinned by UbiITS provides a unique generic and modular foundation for extending the study and analysis of engineering and other business activities from a fundamental and applied research point-of-view as well as a powerful basic technique to support business process improvement.

Chapter 7. Temporal multimodal data synchronisation for the analysis of a game driving task using EEG

Related Publications

A. Sivanathan, T. Lim, S. Louchart, and J. Ritchie, "Temporal multimodal data synchronisation for the analysis of a game driving task using EEG," *Entertainment Computing*, Mar. 2014. doi:10.1016/j.entcom.2014.03.004

Sivanathan, Aparajithan, Theodore Lim, Sandy Louchart, and James Ritchie. "Temporal Synchronisation of Data Logging in Racing Gameplay." Procedia Computer Science 15 (2012): 103–110. doi:10.1016/j.procs.2012.10.062.

Hislop, Matthew, Aparajithan Sivanathan, Theodore Lim, James M Ritchie, Sandy Louchart, and Gnanathusharan Rajendran. "Beyond Simulators: Using F1 Games to Predict Driver Performance, Learning and Potential." (n.d.). – *Accepted Article for Lecture Notes in Computer Science*

Computer games are increasingly being used to study user affect in psychology and behavioural studies. To study a particular phenomenon, such as learning, perception or emotion, often the arbitrariness of real world activities makes for a difficult experimental setup to gather, track and analyse the provenance of data. Temporal data from human bio-signals e.g. electroencephalogram (EEG), electromyogram (EMG), electrooculography (EOG), galvanic skin response (GSR), etc., are often used to study the human behaviour with the reduced attitudinal and demographical influences. Unfortunately, such data are prone to noise and thresholding, weakening the correlations where evidence of temporal influences upon the multi-dimensional and subset operations during in-game activities to produce contextually meaningful information. Having well thought out and implemented game play mechanics will provide a context and immersive experience, by reflecting the temporal phenomena. This case study employs the UbiITS framework to capture temporal multimodal data including bio signals. In particularly it utilises temporal synchronisation techniques provided in the framework to achieve tight time-phased harmonisation across multimodal data channels. Subsequently, it demonstrates the potential for exploiting this multimodal synchronisation to interpret correlations across multimodal data to provide an in-depth understanding of the gaming activity.

7.1. Driving Task and game elements

A study using a driving game play was used to quantify driver performance and skills so as to gain a better understanding of a driver's ability. The intention was to ascertain whether cognitive and motor skills from a driving game transferred to real world driving. The hypothesis that characteristics such as confidence, skills, capacity of learning, etc. in a gaming environment reflect similar skill sets for real-world driving, particularly since these activities align with requirements for Formula (F1) Student driver selection [224].

In this particular study, it is anticipated that learning is achieved via the repetition of the same game activities and a trial and error approach towards task completion. Drivers were asked to drive around the racing track against the clock. Driver abilities vary greatly depending on their real and simulated driving experience (i.e. driving games) and their knowledge of motorsports. The study therefore focused on the learning process rather than performance. The aim being to determine how accurately temporal data can be captured and fused to help document driver performance in the areas of braking point, corner entry, negotiating an apex and corner exit, prior to track days and race-driving tuition.

The data captured in a game session can be classified as context independent and context dependent data [225]. Context independent data is normally external to the game, such as EEG, eye tracking, screen capture, etc., while context dependent data relates to the internal elements of the game such as score, lap time, driving speed, interaction with a game asset or character. Apart from enabling insights into the actual internal affective state of the player [226], context dependent and context independent data permits a more holistic understanding of the combined and interactive user-game system. Consequently, accurate synchronisation of multimodal data streams is critical to avoid parameter skews for analysis. Analysing task based operations (e.g. event related potentials) require precise time measurements where the chronological ordering of events is crucial [227]. For example, if one wants to analyse P300 or N400 components in EEG signal, the accuracy of the synchronisation should be in millisecond range rather than in seconds [228].

7.2. Need for the temporal synchronisation between in-game data and external data

In addition to the game, multiple data capture devices (e.g. EEG capture device, eye tracker and video capture device) also work as detached components in a data logging environment. Devices running on independent hardware or software platforms will naturally run asynchronously. Data captured from these isolated components are required to be synchronized by some means because data streams originating from these components must be temporally aligned to decipher the meaningful information. The temporal alignment can

influence the information extracted in such a way that significant information of an activity is detected, undetected, or falsely detected. For instance, the eye tracking data stream should be adequately aligned with the data stream from the in-game context in order to recognize what in the dynamic scene of the game could have caused the change in the eye data.

Synchronising data streams is conventionally achieved by time-stamping every sample with a central or external clock. However, this is not always a practical solution because of the limitations in accessibility to the internal architecture of games and commodity data capture devices. It is very common to use a data capture device which is usually a proprietary piece of hardware or software; where only the data is streamed out from the device. Although games and device manufacturers provide API/SDK to access the data, access to the core event or the streaming loop is restricted. Therefore it is necessary to comply with the standard data stream provided by the games or the capture devices.

The common approach used in data logging systems is to synchronize all logged channels with one primary channel. Inherently, this primary channel can be a global timing device. Samples from every stream will be marked with a frame number from the primary channel, using it as the index. The TRUE Architecture by Microsoft Game Studios [131] presents a system for recording data for studying user behaviour in a PC software or a gaming console. This architecture looks at streams of data and logs sequences of events along with the timestamp. Captured video is also synchronized with the event timestamps and indexed based on the events. While this demonstrated the efficient retrieval and navigation through large data sets of game play to identify potential problematic areas in a game, fine grained time synchronisation issues are not addressed in this work. In addition to the time stamping, context related events occurring in the game are marked by emitting a unique signal, for example using parallel port byte or a TCP packet. This message is consumed by the data logger and time stamped and/or inserted alongside the other data streams [227]. Emitting events to external software, i.e. to data logging software, needs access to some features of the game via an SDK or API.

7.3. Game - driving task and psychophysiological analysis

Correlating driving performance and psychophysiological data, specifically monitoring the brain activity using EEG can potentially reveal the relationship between driving behaviour and the cognitive state of the driver [38], [229]–[233]. However, the research in the area of distinguishing the above described skillsets from psychophysiological signals is not explicit. Neurometric studies and experiments for brain-based/driven applications are performed in tightly controlled environments (i.e. movement restrictions). It also limits the actual intended

task and it is therefore not always feasible to perform the actual behavioural measurement during the recorded task [234]. Game-driving or real-world driving is essentially a mixture of various visceral and sensorial activities from which the driver has to respond [232], [233], [235]. The brain activity at an instance of driving invariably corresponds to all of these activities.

The conventional practice of neurometric analysis is to ring-fence the study and hence reduce the number of parameters that can affect the experiment. Other activities are recognised as artefacts and the signals relating to these activities are considered as 'noise' in the signals. It might be neither possible to split the driving task into smaller key activities nor reject other activities as artefacts in driving. Consider the task of negotiating a corner at speed, it involves hand-eye coordination and is associated to the driver's confidence level. If a study has been performed to associate the confidence level using the affect state of the driver measured using EEG, separating the movements or visuals processing from task might breakdown the purpose of the study.

Although the action-chain (i.e. epistemic, semiotic and ergotic) relationships [236]–[238] of driving can be broken down into individual elements, the context of behaviour and cognitive organisation cohesion is more than the sum of its parts. The alternative approach is to extract patterns from combined modalities, car-related parameters (e.g. steering activity, pedal depressing, speed of driving, position on the track, etc.) along with nurometrics mainly, EEG.

Consequently, the synchronised data capture of driving telemetry and psychophysiological data is critical for the combined analysis and for finding correlations. Since the outcome of the analysis is highly dependent on the synchronisation of independent data streams, a proven mechanism was required for the data capture and to verify the temporal alignment between data streams. This necessitates re-scrutiny on assumptions made about data capture tools and the whole data stream pathway.

7.4. Data capture setup: temporally synchronised driving game logging and psychophysiological signals

Data was recorded in a driving simulation environment to understand drivers' behaviour, and performance on a racing track. The aim was to monitor a player's activities in relation to mastering the specific corner sequence of braking, corner entry, apex taking and corner exit.

7.4.1. UbiITS implemented data capture setup

The Codemasters Formula 1 game was used as the simulation engine to provide the actual driving experience. Three dimensional visuals of the game were rendered using to the DepthQ

stereoscopic projector onto a power wall with active wearable shutter glasses. Logitech G27 racing wheel and pedals with force feedback system were connected with the game engine as the driving controls. Telemetry data including the car's parameters such velocity, position on the track and engine speed and driver's activities such as throttle, brake and steering angle are streamed out by the game engine via user datagram protocol (UDP) at the rate of 50Hz. Visuals rendered by the game engine were captured as a phase alternating line (PAL) video with 25 frames/second. A video camera was also used to capture the driver and surroundings at the same frame rate. Electrodes were attached to the driver's head, in order to monitor EEG activity. EEG signals were captured at 2048 Hz using Mindmedia Nexus 32 physiological monitoring device. Fig.7.1 illustrates the arrangement of devices and the information flow.

In addition to the driver, an observer was also linked in the data capture setup. The task of the observer was to identify milestones and incidents manually during the game. The observer performed this by pressing predefined keys on a PC keyboard. The observer is also provided with the real-time visualization of the captured EEG data, telemetry and video streams. Although the observer's inputs were synchronously captured with other data channels, it was used to aid a quick post analysis, and had no influence in the real-time driving. For example it was used to point out interesting aspects on every lap, with the hope of future rigorous temporal data analysis and to identify and fix any sensor faults while the experiment was carried out.

Fig.7.1 Driving Simulator data capture setup

Main practical concerns related to the data capture setup and devices are addressed throughout the rest of this section.

7.4.2. Jitter on the data streams

The EEG capture device API delivers the data by means of a call-back function. Having no control over this call-back frequency, other than setting a constant value while configuring, a jitter is observed in this signal (Fig.7.2). This behaviour is expected in a general purpose OS such as Windows and in the universal serial bus (USB) communication channel used by the EEG capture device. A similar observation is also seen in the telemetry UDP packets. Additionally, telemetry stream UDP packets were received approximately at 46 samples/s. This greatly deviates from the pre-set frequency of 100Hz. Network dynamics and UDP packet losses or the simulation engines internal design might have created this deviation. The jitters can be removed by introducing additional First-In-First-Out (FIFO) buffers. The UDP stream's actual frequency is monitored by counting elements on the FIFO in fixed intervals. This observed frequency is used as the real frequency in the analysis.

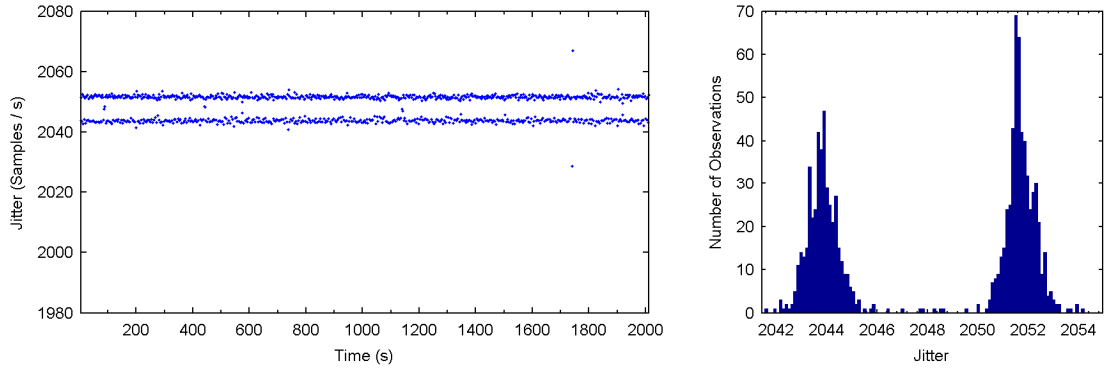


Fig.7.2 Observed Jitter in delivered samples: Jitter observed in the delivered samples, the expected frequency: 2048Hz.

7.4.3. Asynchrony of EEG and telemetry streams

EEG and telemetry streams originate from highly dissimilar sensor nodes. The path taken by EEG samples can be described sequentially such as generated by the hardware, transmitted through the optical fibre, retransmitted by the USB device, received by the API and finally reaches the data capture framework's realm. Since there is no apparent knowledge available to model the path taken by these samples the relative temporal offset between these two streams cannot be estimated directly.

7.4.4. Using a reference device to calibrate stream offsets

An approach to estimate latencies and time offsets of data streams independent of devices' internal architectures was defined in the UbiITS framework (see: Section 4.6.4). For calibration

purposes an in-house built embedded device (Fig.7.4) was used to induce distinctive signatures in individual streams. The technical specifications and the internal architecture of this device are detailed later in Chapter 9. The calibration device was initially programmed to estimate a Round Trip Time (RTT) of a command (i.e. the command to induce a signature pattern) and to receive the respective, symmetric acknowledgement. Measuring RTT can be also affected by the OS timing issues and jitters (see: Section 2.8.1). As a consequence of this, the RTT is estimated using multiple trials (Fig.7.3) in order to find a better estimate.

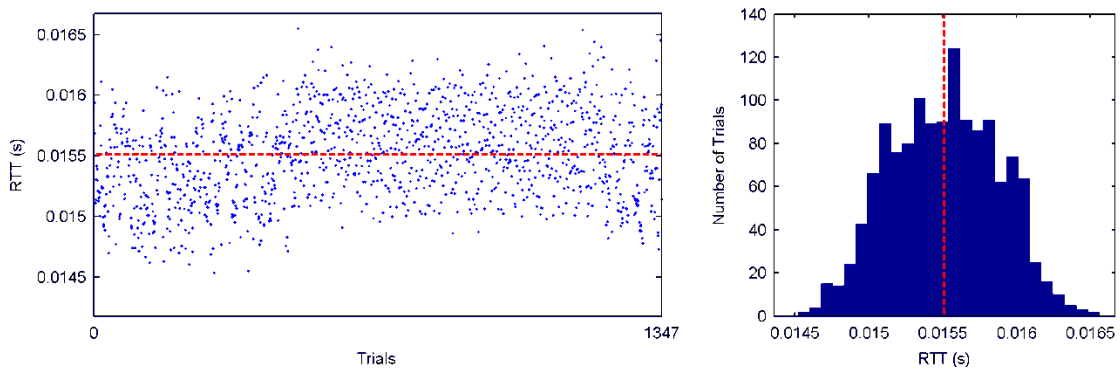


Fig.7.3 Round Trip Time: command round trip time of the custom built device, connected through UART interface at 115200 baud rate was observed as a Gaussian distribution with the mean of 0.015511 and mode of 0.015589

After the RTT estimation, the calibration device was programmed to induce signature patterns. Every stream's offsets were then estimated individually by spawning individual calibration patterns such as inducing spike waves on the EEG sensors using digital pulses, blinking light emitting diodes (LED) on video streams, using gyros attached to the embedded devices to sense the steering wheel rotation etc.

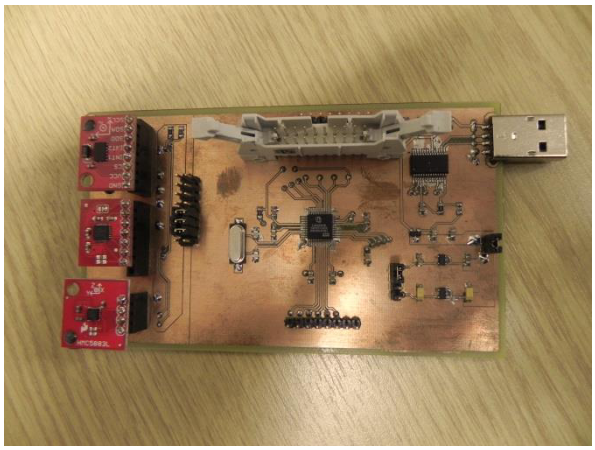


Fig.7.4 In-house built embedded device with inertial measurement modules mounted. (Detailed information about this hardware and its firmware design can be found in Chapter 9)

Temporal offsets for the EEG and video streams were estimated using the method described above and by inducing spikes on their individual data streams. However a slightly different method was used to estimate the offset in the telemetry streams from the game engine.

Inertial data captured using the reference device was used to calibrate the telemetry streamed from the game engine. A one off calibration was performed to measure the telemetry offset. The reference device with the inertial measurement modules was mounted onto the steering wheel and pedals so that the offset between the inertial data stream and individual telemetry streams can be measured. Both telemetry and inertial data streams are examined for the calibration signatures such as sharp turns on the steering wheel and quick depresses on the pedals. Distinctive changes in data streams related to the calibration signatures were recognised and used to evaluate the offset between these streams. Once this offset is measured and using the known latency of the reference device (this is equivalent to the latency comprised in the inertial data stream) the actual latency of individual telemetry channels were evaluated.

7.4.5. Evidence of synchronisation of driving activity and EEG

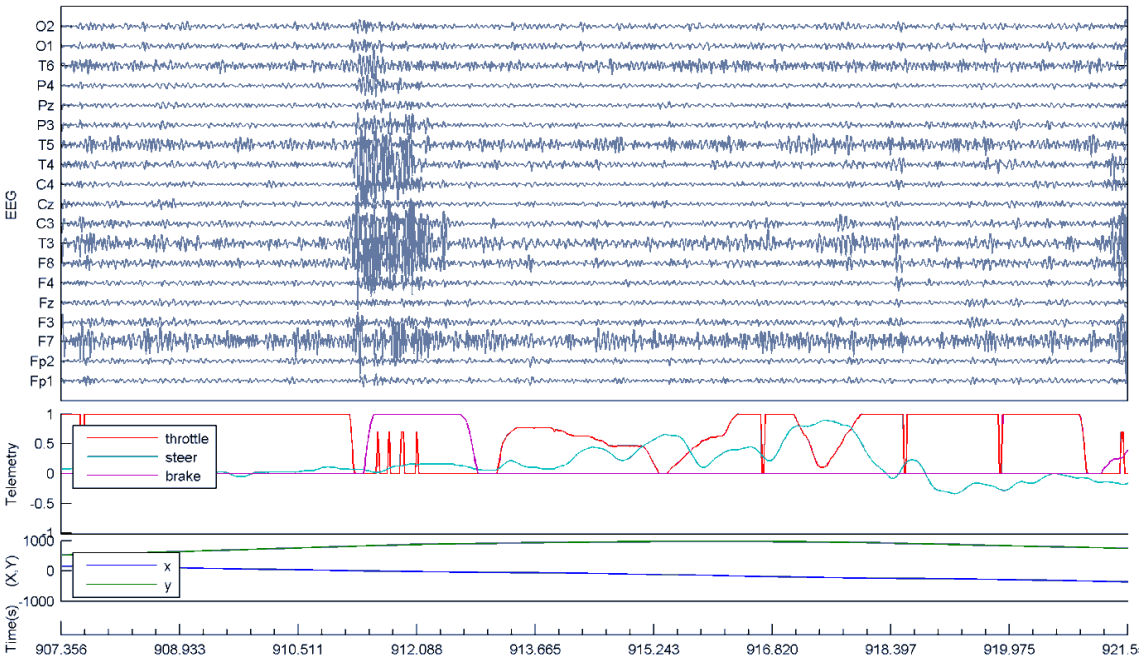


Fig.7.5 EEG and driving telemetry signals synchronised using the technique proposed in UbiITS framework (see: section 4.6.4).

The EEG signals and the car telemetry shown in Fig.7.5 respectively illustrate the correlation between the driver's muscle activation of the actuations on the game controllers (i.e. steering wheel and brake and acceleration paddles). EEG signals were filtered using Butterworth, band pass (20-50Hz) filter so that the muscle artefacts are clearly visible and hence the

synchronisation can be validated. Muscle artefacts are typically filtered out in an EEG analysis and disregarded but this information is effectively used here for the purpose of synchronisation.

Muscle activity is a common artefact which can be manually observed in EEG signals, therefore it is used here to provide evidence and validate the synchronisation of the captured signals. Accuracy of the temporal alignment of the EEG and driving telemetry can be visually examined using muscle actuations which correspond to a driving activity, i.e. driver is changing over from full throttle to brake and subsequently performing a turn in an apex in the track. It should be noted that in this case muscle artefacts present in the EEG data stream were exploited to validate the synchronisation, and hence the signals were filtered to enhance the muscle artefacts. Nevertheless it is common to filter out muscle activities while performing a comprehensive EEG analysis, however the concept of synchronisation will remain unchanged.

7.5. Brain activity and driving patterns

Having addressed the synchronisation-related issues described and with temporal alignment of captured driving telemetry and EEG data reasonably justified, the methodology was used to study the driving pattern of 14 drivers, including 7 gamers with more than 500 hours of formula 1 gaming experience, 2 real formula 1 drivers and 4 rally drivers.

The main aim of this study was to demonstrate how synchronised data logging and multimodal data fusion, i.e. multiple data streams such as psychophysiological signals, game and telemetry can be useful in documenting the process of learning. In this case, learning to corner at speed; braking, corner entry, apex and corner exit. Synchronised driving activity data provides the necessary information for documenting the learning process from a purely numerical perspective. Such data also allows for determining learning and exploration patterns regarding a task.

7.5.1. Experimental setup

The Silverstone circuit was selected for the study with each participant driving the Red Bull Racing RB7 Renault car with no other cars on the track. The car was configured for automatic gear shifts, 197 degrees of steering angle and all other assisted driving controls such as breaking, stability control, etc. disabled. To provide a more immersive environment full cockpit view, force feedback steering and stereoscopic three dimensional rendering were enabled. Lap count, lap time and shortest lap time were displayed on the screen while all other information displays, (e.g. track position, track guidance marks, comments) were turned off. Fig.7.6 shows the driving environment and the rendered 3D visual.

All drivers were trained on the Melbourne Grand Prix circuit before they were allowed to enter the Silverstone circuit. The training was to familiarize the driver to the controls, the car, and adapt themselves to the simulated environment, (i.e. depth perception, motion sickness and steering response etc.). Once the driver felt competent, they were asked to complete as many laps as possible within 30 minutes in the test session on the Silverstone circuit.

EEG data was collected in a 10-20 system using an electro gel cap. Drivers were strapped to the driving seat using seat belts and a head-neck restraint setup similar to a HANS device [239]. These restrains emulated an environment similar to a real F1 car and helped to restrict intense body movements, hence reducing severe artefacts in the EEG.

Fig.7.6 F1 driving simulation environment with the 3D power wall.

Driver activity over a specific region on the track (Fig.7.7) was selected. The 'Stowe' corner was chosen as it is a high-speed corner with a long approach straight leading into a large radius curve. It was anticipated that drivers will contemplate risks as they learn to deal with this corner. Additionally, Stowe tests the confidence and the attention/concentration of the driver since maintaining a high entry and exit speed critically impacts on the overall lap time. The A telemetry capturing module was built to automatically generate a marker in real-time whenever the car enters and exits Stowe. The automatic generation of markers allows instantaneous extraction of temporally aligned data, i.e. ready to use data segments during or after the driving session.

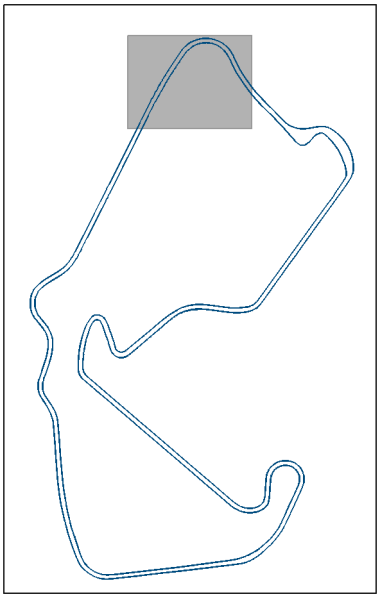


Fig.7.7 Formula 1, Silverstone track [240] and region of interest for the study, i.e. 'Stowe' corner.

7.5.2. EEG processing

EEG analysis was performed in both time domain and frequency domain, mainly based on the signals from frontal lobe electrodes (F3, Fz and F4). Signals were filtered using 4th order Butterworth filter, using hi-pass to remove the DC component and low-pass to remove the high frequency noise. Actual cut-off frequencies were selected according to the specific analysis performed, i.e. alpha band 8-12Hz and Beta 13-21Hz [241, pp. 112–120]. Thereafter the filtered signal was transformed into the frequency domain to observe the power of the frequency power spectrum. The frequency transform was performed in 1s windows, with a window stepping of 10ms. The frequency transform was scaled based on the length of window (number of points) to conserve the signal power; although more importance was given to the relative variation of the frequency spectrum than absolute values. Various representations of the frequency power values have been presented throughout this section. The colour map used to represent the frequencies varied from blue to red corresponding to the minimum and maximum frequencies in the considered time span.

7.5.3. Artefacts and noises

A driving task involves various activities that could influence the EEG recording, such as muscle activities in the head region, eye blinks, etc. Body and head movements also cause slight movements in the EEG cap, affecting the impedances in the electrodes, consequently introducing many noise sources in the signal. Artefacts occur more frequently in driving compared to a controlled experiment, e.g. an ERP experiment with limited predefined events and responses. The standard practice is to automatically or manually detect and reject artefacts [242]. However, doing this for an EEG signal collected during a driving task would

result in highly fragmented strips of signals unsuitable for continuous brain activity analysis [36], [243]. Event locking is a common methodology used in EEG studies of signals before and after the event. Thereafter the signals will be averaged over many events and/or participants [242], [244]–[246]. The difficulty with this approach is isolating and distinguishing a discrete cognitive event in a driving task.

An experimental system built with consumer game equipment and EEG capture is limited in many ways compared to an exclusively built laboratory setup used in psychophysiological studies, e.g. cost and quality of the equipment, signal to noise ratio, interferences by external sources and activities, etc. Body movement is a significant source of noise particularly since it is almost impossible to maintain a 'rigid' posture or abstain from body sway. These are uncontrolled parameters as result of combined game immersion and motor reflex as in a real-world racing environment, whereas parameters are carefully controlled in a scientific experiment so that one specific concept can be isolated and studied independently. The contrast here revolves around using a force feedback steering wheel and pedal box as opposed to a game pad with thumb-stick controls. Therefore, the approaches followed in a controlled environment are not always practical in a more realistic environment.

As a consequence, rather than establishing a unique signal or pattern, it is preferable to identify the trends relating to action-chain relationships in the EEG data. Potentially driving activity can be explained more reliably and accurately by one or more sequences of abstract action-chain relationships. Therefore it is important to know what happened at a specific moment in the environment to understand and interpret the EEG. The basis developed by the tight synchronisation provides and enables the correlation of recognisable actions performed by the user at any specific time.

7.5.4. Interpreting the 'noisy' EEG using synchronised telemetry

The possibility of revealing some useful information from artefacts is now considered here. Instead of removing signal components, which are deemed to be artefacts or noise in conventional EEG, they are retained. For example, signatures of white noise which appears in all frequencies could represent a muscle artefact or disturbance caused by a movement in the electrodes, cap, cables, etc. (Fig.7.8). White noise is identified by vertical lighter lines in the frequency spectrum, some of which are associated with the driver's physical activities on throttle whereas horizontal lines represent alpha activity. Therefore such a white noise may be considered as a useful indication of the driver's body movement and verified further if it also corresponds to a driver activity captured in driving telemetry. Observing the continuous variation in alpha power (horizontal colour changes) it can be visually justified that every rise in

alpha is not due to an artefact and hence the data still contains usable information, therefore it does not need to be disregarded completely.

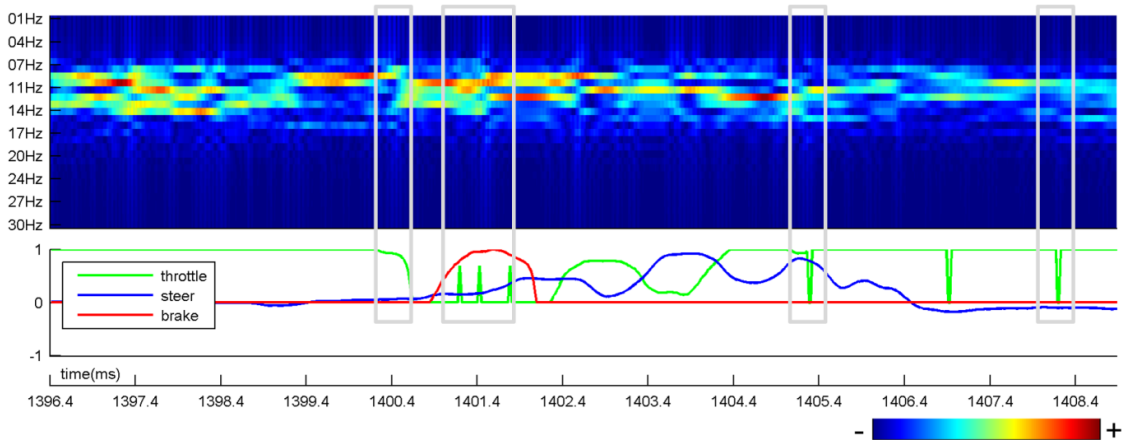


Fig.7.8 EEG frequency power spectrum, signals were filtered using the pass band of 8-13Hz (alpha band) before transforming into frequency domain

7.5.5. Potential temporal correlations

From an observational perspective, mistakes made by drivers and the type of alpha and beta activities immediately prior to or during mistakes being made were investigated. Equipped with such data it is then possible to further identify the causes for mistakes associated with attitude parameters such as concentration (Beta) or attention (Alpha). The resulting analysis below suggests several cases where the synchronised data from EEG can further document an observation and potentially provide better feedback towards learning or skill acquisition.

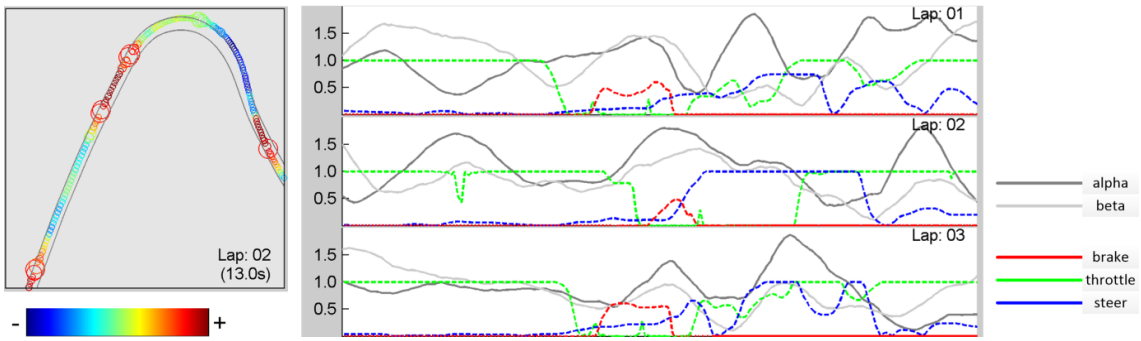


Fig.7.9 Left: Alpha power mapped on to the car's path. Change in throttle is mapped to the radius of the circles. Right: Alpha Beta, brake throttle and steering angle are plotted against the distance

In Fig.7.9, the driver makes a mistake at the breaking point; brakes too late, thus missing the corner's entry point, apex and corner exit. The cause of the mistake is clearly seen from the telemetry record and trajectory of the car in lap 2 compared to laps 1 and 3. A possible interpretation of brain activity correlated to the incidents in the driving is given below.

It can be speculated at this stage that alpha monitoring could provide information as to why a driver might have missed their braking point. Research has shown EEG alpha power fluctuations to correlate with processing of the visual network and between the visual cortex and the rest of the brain. Interestingly, these effects are unique for the alpha band and not observable in other frequency bands. [247] The braking zone is highlighted in red, meaning that the driver is emitting high alpha thus appearing quite relaxed and still possibly performing mental calculations, working with memory [248]–[251]. Upon missing the breaking point, the alpha reading shift towards low alpha (blue) and full attention to the task at hand. It can be speculated that this could illustrate the moment in which the driver realises that he/she is not going to make the corner entry or the apex and possibly run off the track. While speculative, the shift of activities in the EEG bands appears to match the expected experiential sequencing of events on track. In this context, the later part of the corner is particularly telling as the driver is focusing his/her attention towards re-positioning the car on the track. Fig.7.10 illustrates further examples from a novice driver, highlighting the alpha attenuation when coming close to the kerb or overshooting. Similar interpretations were derived by performing cause and effect analysis [252] for all 14 drivers.

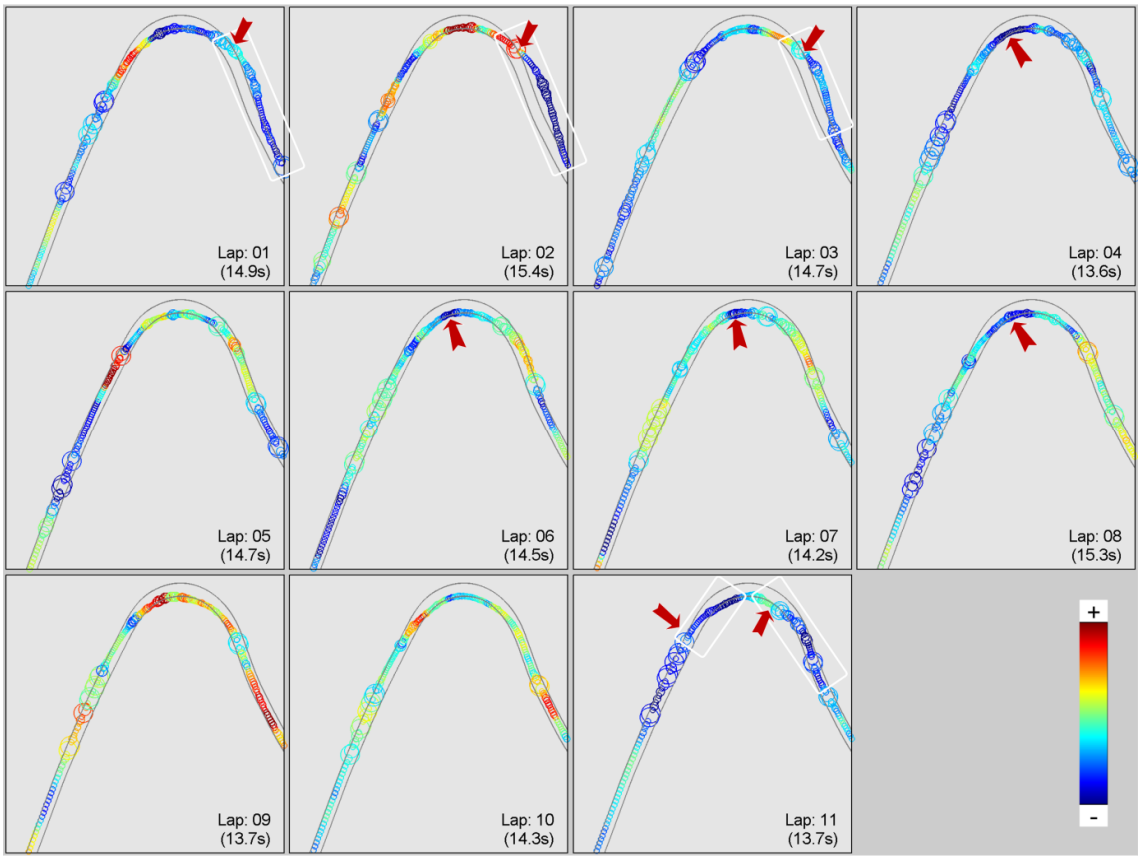


Fig.7.10 Power of alpha frequency is mapped on to the car's path. Change in throttle is mapped with the radius of the circles. Arrows indicate car coming adjacent to the kerb or an overshoot.

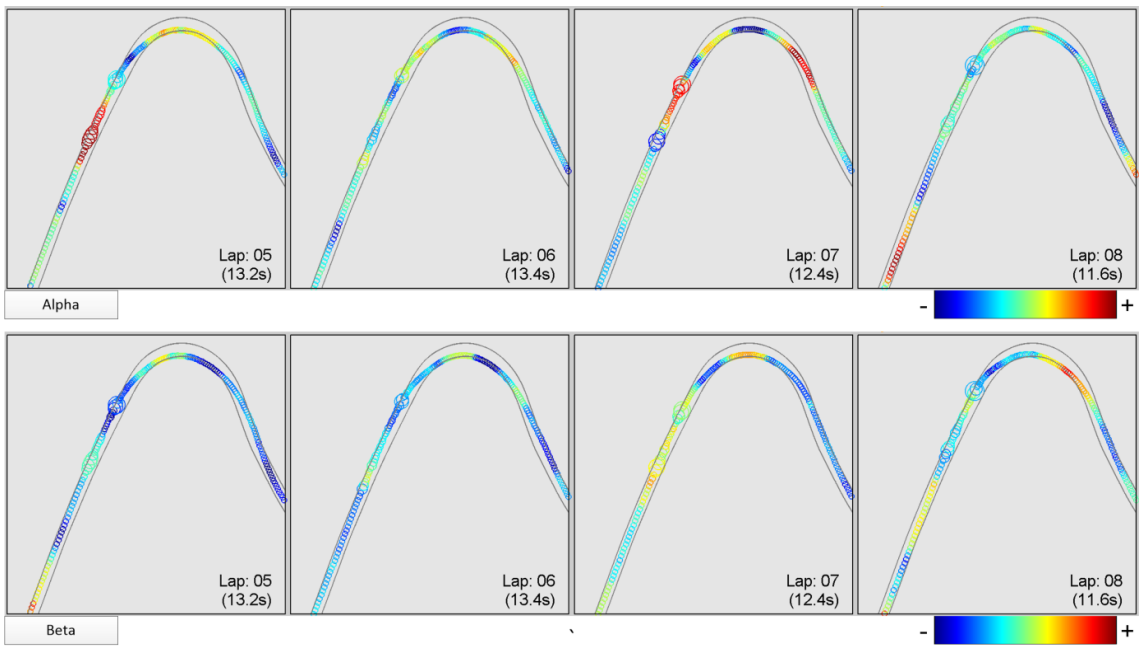


Fig.7.11 Alpha and Beta power changes are compared for a driver in 4 consecutive laps. Change in breaking is mapped on to the circles' radius.

While high alpha is interpreted as a state of still brain, i.e. relaxation or performing internal mental calculations, working memory [248], [251], beta represents a busy activity, anxious thinking, or actively concentrating [249], [253]–[255] process. Fig.7.11 shows the drivers brain switching between alpha and beta activities. Low alpha power was observed when the driver overshoots the track then he/she regulates the throttle and corrects the path back to the normal position. This is justified by the notion of alpha attenuation i.e. increased attentiveness [248]–[250]. The example in Fig.7.12 illustrates a situation in which both alpha and beta activity can be combined in order to further explain a mistake and cognitive load in compensating the error. The switch between alpha and beta frequencies in the apex can be observed, i.e. the driver switching between relaxed, possibly internal calculations, working memory modes (high alpha) to attentive (low alpha) [248], [251], and busy, anxious thinking, active concentration (high beta) modes [253], [254]. In this case, the error occurs at mid-corner while negotiating the corner's apex. At this stage of the task the driver's attention should shift towards the corner's exit and target the white track demarcation line on the side of the track for maximum exit speed.

In this particular case, there might be several reasons as to why the driver has run off the track, e.g. carrying too much speed in the corner and/or misjudge the corner exit point. Fig.7.12 however is suggesting that the driver might not have been as attentive as he/she should have been towards the apex of the corner. High alpha towards the apex of the corner would suggest that the driver is quite relaxed at this point. Low beta suggests that the driver in

this case is not showing high levels of concentration (blue). Once off the track, alpha levels reduce, thus suggesting that attention has fully returned while beta activity slowly rises (yellow/red) towards re-joining the track.

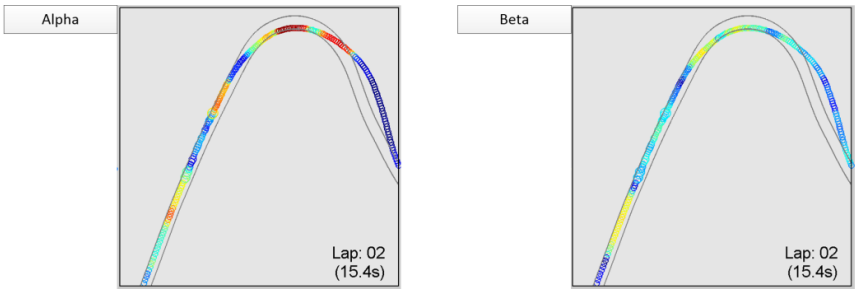


Fig.7.12 Alpha and Beta power changes when overshooting the track

While this approach cannot at this stage be used as a definite and conclusive mechanism for interpreting player performance, it can be used in conjunction with other game-related performance and logged activities so as to further determine the reasons motivating player behaviours and enhance the quality of feedback that can be offered to the player.

7.6. Reflections and opportunities

The work presented highlighted the potential of interpreting the driving based on EEG and in game data. The difficulty in tightly controlling the environment and ring fencing a particular activity in the driving task creates difficulties in employing conventional neurometric approaches. Consequently it is necessary to choose a flexible approach of using time synchronised multimodal approach to interpret the data. Cause and effect analysis have been performed manually to describe patterns in the EEG power spectrum [252]. Current and future work will focus on automatic classification and documentation of the driving activity. Differences in previous experiences in gaming or driving task and driver individualities create a challenge in building up a large dataset. The reliability and performance of automatic methods rely on finding suitable statistical methods and adequate amount of training data for classifiers. Using the time synchronisation techniques presented the future direction of the work leads towards a real-time, closed-loop system that can feedback into the game or to the driver to influence driving experience

7.7. Conclusion

This case study employed the novel and generic technique proposed in the UbiITS framework to synchronize independent data streams captured in a gaming session. The technique described neither depends on nor needs to have knowledge about the internal architecture or the core timing loop of the game to measure the end-to-end timing inconsistencies between

the environments (i.e. where real-world user activity happens) and the data capture endpoint (i.e. the data is being available to capture along with the temporal information).

This time phased recorded video, driving telemetry and EEG data, demonstrate the accuracy of synchronising these multimodal data streams for neurometric analysis using the UbiITS framework. There are no automatic mechanisms built in the game architecture or in commercial data capture systems for such multimodal data synchronisation and data fusion. As an alternative, ad-hoc solutions are available but this requires users to configure a reference device in order to calibrate the recordings. Selecting a suitable calibration approach depends on the nature of the application and the corresponding data stream. Using light flashes for video capture, electromagnetic induction on cables for EEG capture and calibrating the steering wheel telemetry using the inertial measurement units are the exemplars. Considerations are needed for data stream offsets and the jitter parameters to compensate the asynchrony and the nondeterministic characteristics throughout the data stream's pathway. This case study has provided evidence for the temporal synchronisation to justify its claim about its accuracy. In addition to the visually observable associativity across signals (Fig.7.5) the demonstrated possibility of exploring temporal correlations of internal human activities such as brain status (Fig.7.10, Fig.7.11, Fig.7.12) further validate the accuracy of the synchronisation.

The temporal synchronisation across multimodal channels was exploited to create in depth interpretations of the in game activity. This approach provides fresh opportunities for using commercial of-the-shelf games to study user behaviour in-relation to the in-game data logging. Consequently, UbiITS enables the vast majority of games not intentionally designed for research to be used for scientific and methodological research. The development of such a generic technique for temporal synchronisation opens up a wide range of possibilities in the areas of psychology, game studies, cognitive sciences and experiential-based multimodal studies. This technique is beneficial to the Serious Games community in the sense that it offers a quick method for a synchronised real-time documentation of learning activities by combining explicit and implicit data gathering. From a multi-modal perspective, this approach presents the benefit to develop a flexible and fast approach to experimenting with complex experimental designs by significantly reducing the complexity of experiment technical implementation.

Chapter 8. Mirror Game - A Neurofeedback protocol using a sophisticated interactive video game to improve social responsiveness in children with ASD

Attestation

This case study is about a system implementation named Mirror – a Neurofeedback system currently being used for core neurophysiological research by collaborative research teams in the Department of Cognitive Science, University of California, San Diego, USA and the Department of Psychology, University of Graz, Austria. This demonstrates a technological readiness level related to this work and that is beyond the laboratory. The following is an attestation about UbiITS provided by a collaborative research team in the Department of Cognitive Science, University of California, San Diego.

"Physiological studies always encounter the problem that they need very specific markers in the data to link a physiological reaction to a specific event. Most commercial games do not send such specific markers and can therefore hardly been used for physiological studies. The framework [UbiITS] allows connecting physiological measurements with a game and handles the trigger sending. Moreover, the timing of the separate software (i.e. game and recording device) is synchronized within the framework. This is especially important for applications that provide the user with real-time feedback such as in Neurofeedback studies. With the framework, up to 8 control channels can be defined for real-time control, it is easy to add devices such as a webcam for certain measurements and the different devices/software can be run on the same or different computers. Another very useful feature of the framework is that it indicates the buffer size as well as the number of data samples and markers transmitted, so the connection and data transfer can be supervised continuously." - Elisabeth Friedrich, May 29th, 2013

Related Publications

E. V. C. Friedrich, N. Suttie, A. Sivanathan, T. Lim, S. Louchart, and J. A. Pineda, "Brain–computer interface game applications for combined neurofeedback and biofeedback treatment for children on the autism spectrum," *Front. Neuroeng.*, vol. 7, p. 21, 2014. doi: 10.3389/fneng.2014.00021

Elisabeth V. C. Friedrich, Neil Suttie, Aparajithan Sivanathan, Theodore Lim, Sandy Louchart, and Jaime A. Pineda, "A Neurofeedback protocol using a sophisticated interactive video game to improve social responsiveness in children with ASD," in *Society for Neuroscience, 2013*, New Orleans, LA, 2013, p. 181.15/AAA24.

8.1. Background information [1]

Children with autism spectrum disorder (ASD) show deficits in social and communicative skills such as imitation, empathy and shared attention as well as restricted interests and repetitive patterns of behaviours [258]–[260]. These deficits substantially impair social interactions and may have a basis in motor, emotion and/or cognitive deficits. The aim of this study was to elicit, enhance and train appropriate motor behaviour and cognition as the basis for emotional reactions in social situations for children with ASD.

Dysfunction in imitation, specifically in the mirror neuron system, is speculated as one of the underlying causes for these symptoms in people with ASD. Neurofeedback was shown to reduce the symptoms in children with ASD by self-regulation of brain rhythms [261], [262]. Peripheral physiological activity such as the heart rate is closely linked to neurophysiological signals as well as associated to social engagement [263]. Therefore, this study proposed as effective a combined approach of highlighting the interconnected activities of brain, body and behaviour in children with ASD. This study was focused on establishing an innovative system that provides feedback corresponds directly to the underlying significance of the trained signals as well as to the behaviour being reinforced.

This system was primarily based on the use of a specifically-designed interactive video game and neurofeedback training (NFT) as an operant conditioning methodology to gain control of specific brain dynamics. The assumption being that modulation of the mu rhythm (8-13 Hz) can be trained with NFT and will reengage the mirror neuron system (MNS). The MNS provides the neurophysiological basis for imitation and has been found to be dysfunctional in individuals with ASD. Furthermore, a linkage between the regulation of heart rate and the social engagement system was proposed. Therefore, peripheral physiological reactions to emotions in social situations are recorded in this study.

8.2. Interactive game and real-time feedback[2]

Children with high functioning ASD and typically developing controls played a video game while electroencephalography (EEG) and other peripheral physiological measurements were recorded (e.g. muscle activity, skin conductance, heart rate, respiration). The children are trained to allow a series of social interactions to take place in the game. Specifically, the child's game character engages in the imitation behaviour of facial emotions. During this social interaction game, children are rewarded if they show a specific neurophysiological reaction i.e. modulation of the mu rhythm. This connects the neurophysiological basis for imitation

[1] Information in this section is extracted from: [256], [257]
[2] Information in this section is extracted from: [256], [257]

learning and MNS with appropriate imitation behaviour in social situations. Therefore, greater improvements in behaviour and cognition in the children with ASD were anticipated and this could be evaluated with various tests before and after the NFT.

The ultimate goal of this study was to teach children with ASD to appropriately respond to social situations while numerical data is being gathered using the developed system. As social interactions build the fundamentals of human lives, an improvement in social interactions could strengthen relationships with the family, facilitate social relationships as well as academic interactions and potentially improve the health and well-being of individuals with ASD across their lifespan.

8.3. Real-time closed loop feedback and synchronisation

The predominant technical complexity in this study was to integrate the data capture devices, processing modules, storage modules and the interactive game in real-time. The UbiITS framework was implemented to integrate these diverse multimodal tools and provide closed loop feedback in real-time. Fig.8.1 shows the deployed centralised system based on the UbiITS framework incorporating data capture hardware, real-time processing tools, storage modules and the data flow. Basically the user interacts with the mirror game while the game adapts its dynamics according to the biofeedback provided in real-time.

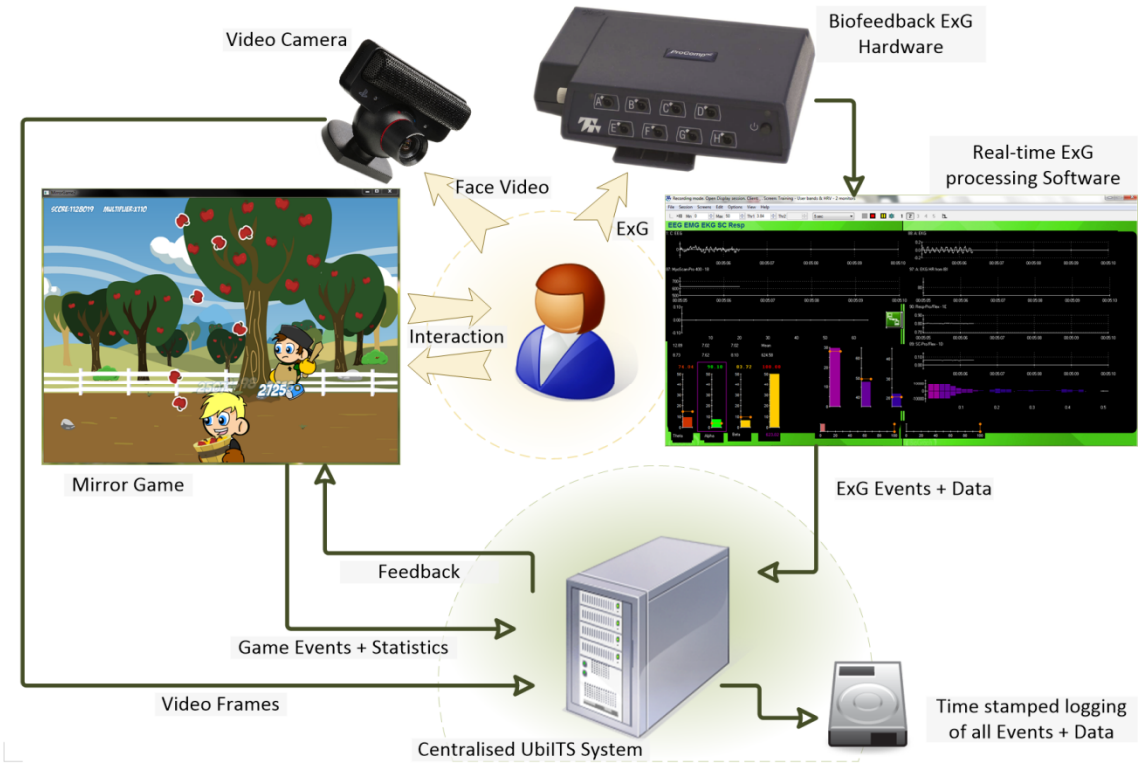


Fig.8.1 UbiITS framework based centralised system deployment with devices, processing tools and data flow

Biofeedback channels were connected to the subject using the Thought Technology biofeedback hardware [264]. Multiple neurophysiological (EEG signals) and other peripheral physiological signals, e.g. EKG, EMG were captured by the hardware and processed in real-time using the Biograph [265] psychophysiological signal processing software. Capturing up to 8 channels was supported by the hardware and the associated API. The software generated discrete events (e.g. threshold met, pattern matched, etc.) in real-time, besides filtering and processing. A centralised system was built to filter these events and route them to the game. The game adapts its dynamics specifically in accordance to these events. In addition to the events feedback to the game, it also generated events associated with in-game activities performed by the user. While the main requirement of the centralised system was to integrate the independent tools, synchronise and support data transport in real-time, it was also required to store all data for post capture analysis. The following data streams and synchronised events were used for this purpose.

1. All events generated by the game were time stamped and stored.
2. All events generated by the psychophysiological signal processing software were time stamped and stored.
3. 8 data streams from the psychophysiological signal processing software were stored.
4. A data stream from the game comprising periodic in-game statistics was stored.
5. A stream of video frames of from a video capture device used to capture the facial activities of the patient was encoded and stored.

Fig.8.2 shows the physical setup of the system with a patient participating in the experiment. This system deployment comprised three networked Windows-based PC platforms. The centralised event routing, video encoding and data storage was performed in one PC while the game and real-time psychophysiological processing were running in two other separate PCs.

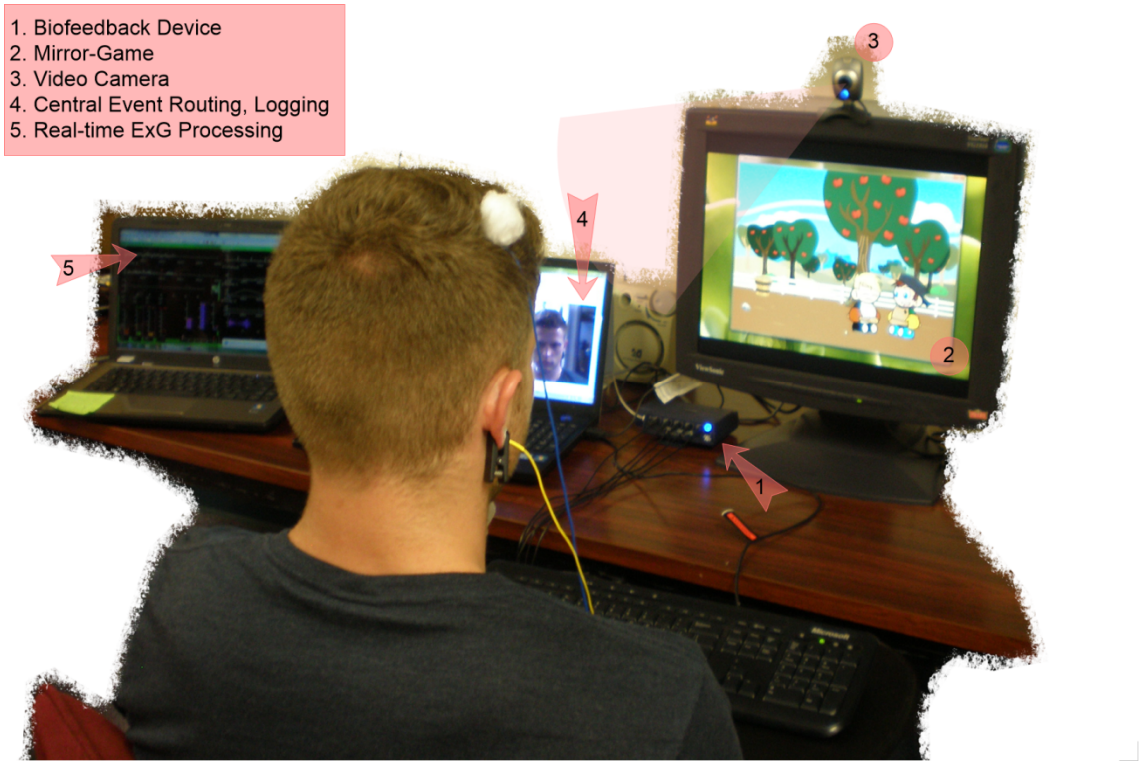


Fig.8.2 Physical setup of the mirror game closed loop feedback system

8.4. Closed-loop, low latency connection of the user space with computational space

The main issue with real-time, closed-loop systems involving human interaction is that the real-time interaction is affected by the latency. Extended latencies can influence the experience and has the potential to skew the results. For instance, in this DCIS biofeedback events needed to be processed in real time, transported and delivered to the game with minimal latencies so that highly responsive interactions can be performed by the game. The DCIS deployment employing the UbiITS framework achieved these requirements using the techniques listed below.

1. Optimising the buffering in individual channels to provide a balance between jitter management and latency.

2. Distributing the computational load across multiple deployments on independent platforms while interconnecting them for synchronisation and data transport purposes.

3. Enabling computations to be performed in multiple computational threads within one platform such that baring a load balance between high priority low latency tasks, i.e. event routing and computationally intensive background tasks, e.g. video encoding.

4. Real time monitoring of the data flow as a mean of recording the dynamic characteristics of the system.

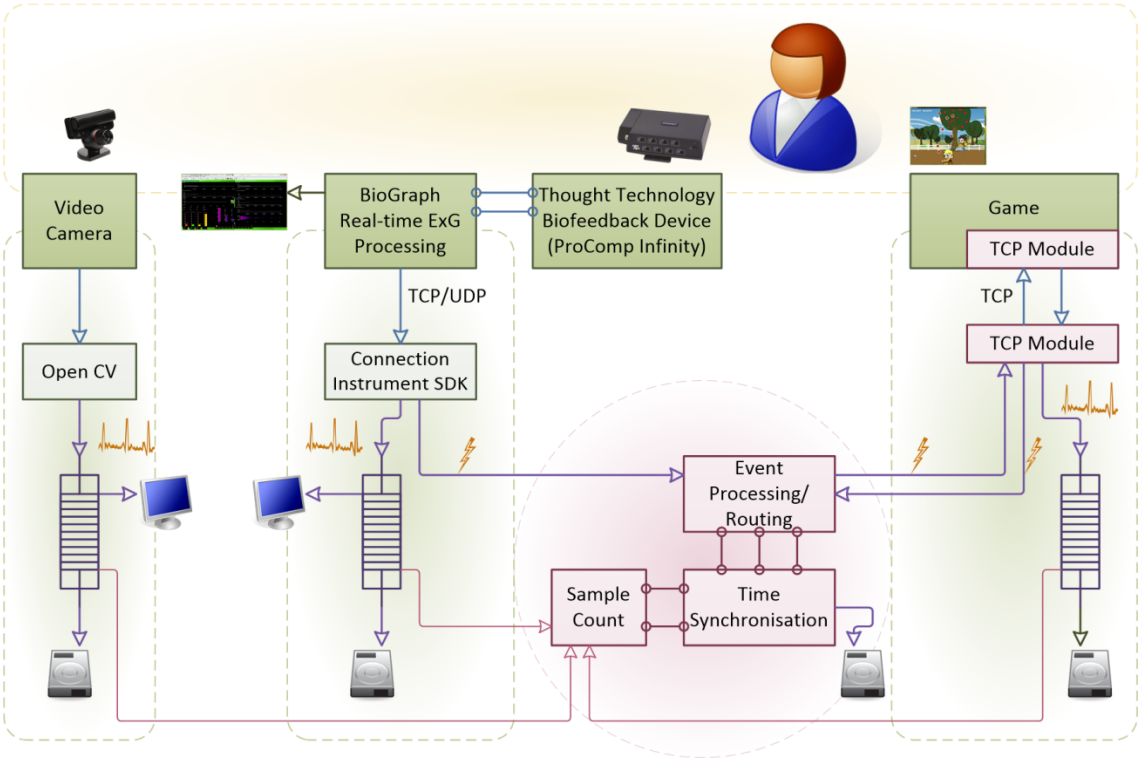


Fig.8.3 UbiITS framework - Real-time closed-loop connecting of physical space and computation space

A brief illustration of important components and various modules based on the UbiITS framework is shown in Fig.8.3. This system essentially demonstrates how the physical space where the interactions were performed was connected to the data processing and computational space, using the UbiITS framework based modules. Subsequently, Fig.8.4 shows an example of a data channel monitored online. In this case the online monitoring of video frames enabled precise synchronisation with spontaneous events. This has been used for an analysis focused on micro-saccade from the facial video where a tight temporal synchronisation with in-game activity was essential. Similarly Fig.8.5 shows synchronised in-game events with psychophysiological signals for a post capture analysis.

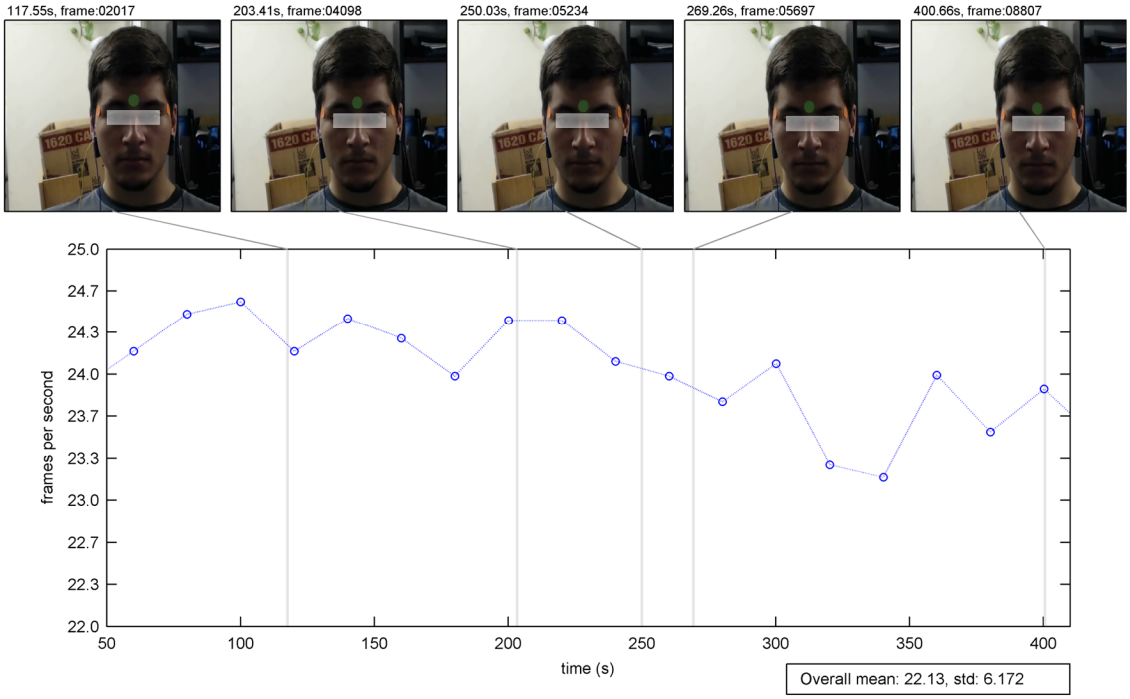


Fig.8.4 Online monitoring of video frames used for an analysis focused on microsaccade from the facial video. Samples were synchronised based on the automatic forced syncing intervals technique proposed by the UbiITS framework

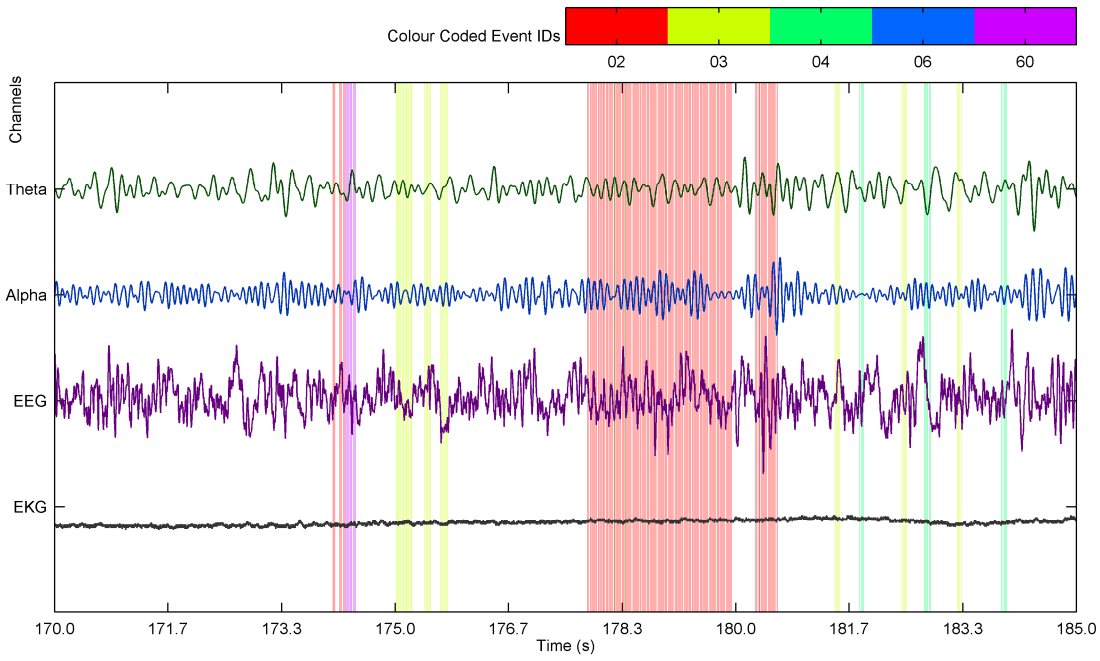


Fig.8.5 In-game events synchronised with psychophysiological signals for a post capture analysis. Events are colour coded for visualisation.

8.5. Conclusion and summary

The case study Mirror - a real-time biofeedback game demonstrated the application of UbiITS framework to solve real-time, closed loop problem. The framework successfully provided the fundamental low-level architecture for addressing the technical requirements of multimodal device integration, data transport across independent nodes the system and tight temporal synchronisation of data channels. This application outlines a higher level closed loop biofeedback system operating on top of the foundation provided by the framework architecture.

The DCIS architecture was specifically designed and customised for neurophysiological studies. This particular system implementation is currently being used for core neurophysiological research by collaborative research teams in Department of Cognitive Science, University of California, San Diego, USA and Department of Psychology, University of Graz, Austria. Beyond the fundamental technical capabilities of the UbiITS framework, this demonstrates the UbiITS framework's stability and readiness to be used as a product by external organisations.

Chapter 9. Sensor prototype and hardware implementations

The previous application case studies used mostly proprietary hardware and software as tools for sensing and interacting with the physical space. These devices were wrapped inside a UbiITS framework device module designed to accommodate a specific device or a family of devices into the system. Generally, once a module is designed for a device by the implementer it can be used any number of times and ported across different systems using the UbiITS framework. All of these modules use APIs and SDKs provided by the device manufacturer as the only means for integrating the device with the framework. Separate from accommodating proprietary devices into a system this chapter demonstrates the capability of the framework to be implemented directly on a hardware platform. In doing so it eliminates the wrapper interface to hardware. As an exemplar of an embedded architecture this chapter presents a custom sensor device with built-in framework components that is readily usable in a system using UbiITS.

9.1. UbiITS built-in hardware

Integrating modules carrying the conceptual functionalities defined by the framework directly on to device firmware enables a device to be highly interoperable and capable of being used efficiently and effortlessly. A device having the functionalities required by the framework readily installed makes it immediately compatible with the rest of the system. This implies that a device having built-in framework modules can start to operate immediately as a plug-and-play device, given that a means of connectivity to the rest of the system is provided e.g. TCP, Serial, etc.

In addition to the plug-and-play characteristics, these devices can be anticipated to provide other benefits such as low latency and much tighter synchronisation. For example, samples from a device can be synchronised by the UbiITS framework only after they have initially been delivered inside a framework module, i.e. usually the wrapper module of a device. Samples inside a propriety hardware and an associated API are generally out of reach of the framework. For this reason the UbiITS relies on FIFO buffers to synchronise the samples (Section 4.6). If the samples are synchronised directly onto the hardware platform where they are captured, the sample latency and the effect of jitter in the connectivity path can be reduced. These hypotheses of a UbiITS built-in hardware related to the low latency and tight synchronisation characteristics will be testified later this chapter.

9.1.1. UbiITS built-in embedded sensor

A sensor device was developed from scratch from fundamental electronic components to demonstrate the concepts of integrating the UbiITS modules in the firmware architecture. This

device is an integrated sensor with a 3-axis digital gyroscope, accelerometer and compass, additionally having 8 channels of auxiliary analogue to digital conversion (Fig.9.1). It can be used primarily to measure orientation and inertial effects, for example in the previous F1 case study to measure physical steering movements.

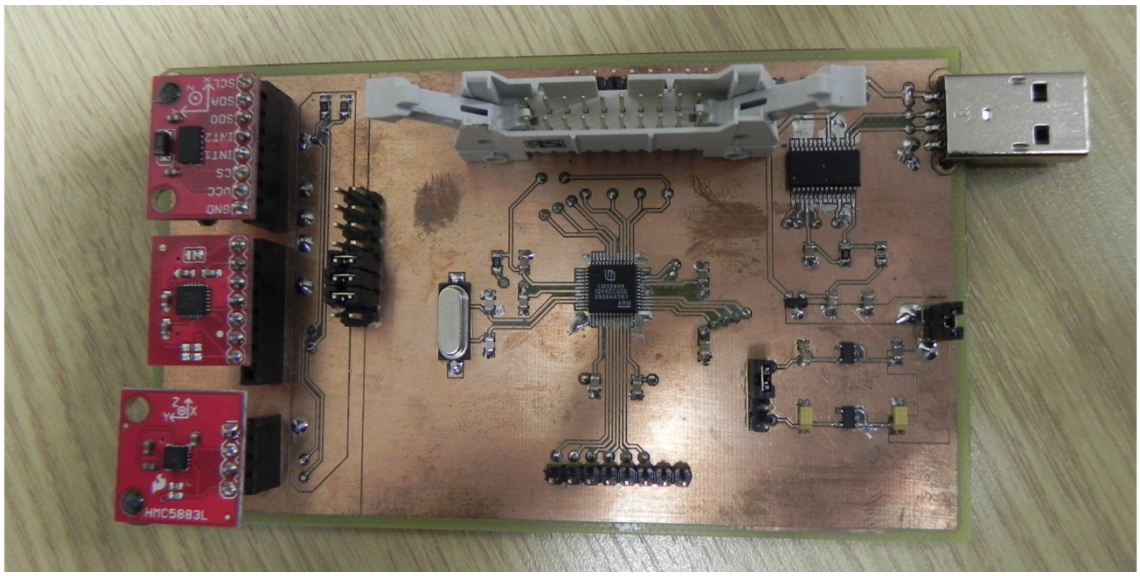


Fig.9.1 UbiITS built-in embedded sensor

9.1.2. Hardware architecture

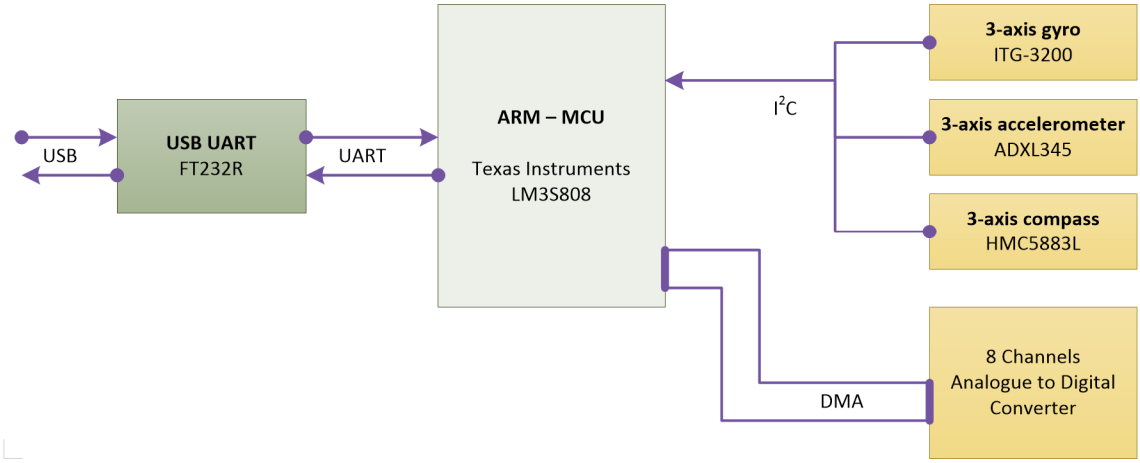


Fig.9.2 Hardware architecture of the integrated sensor device. Detailed information about the actual electronic schematics and printed circuit board layout can be found in Appendix 1

A Texas Instrument 32bit ARM microcontroller unit (MCU) was used as the central piece of this integrated sensor in which the framework modules were implemented. It can be connected to a PC via a USB connection emulating Serial connectivity. Fig.9.2 illustrates the hardware architecture and the components used. The three orientation sensors (inertial and compass) were connected to the device via an Inter-Integrated Circuit (I^2C) bus. The auxiliary analogue to digital conversion (ADC) was carried out by the MCU using the integrated ADC peripheral

which was accessed via the direct memory access (DMA). The MCU was programmed using C language using IAR tool chain [266].

9.1.3. Interrupt driven firmware architecture

The device firmware architecture was built directly on the "bare metal", i.e. no operating systems were used. Having the architecture directly on the bare metal removes the operating system's nondeterministic task scheduling and timing artefacts and consequently the timing behaviour of the device is completely deterministic. An interrupt driven firmware was built integrating the UbiITS framework. Prioritised pre-emptive scheduling was chosen for this firmware architecture since this task scheduling policy is suitable for the highly strict timing requirements of a real-time application. Additionally this scheduling policy was supported by the MCU and its ARM architecture. Fig.9.3 illustrates the main components of this firmware and the tasks concurrently executed in real-time.

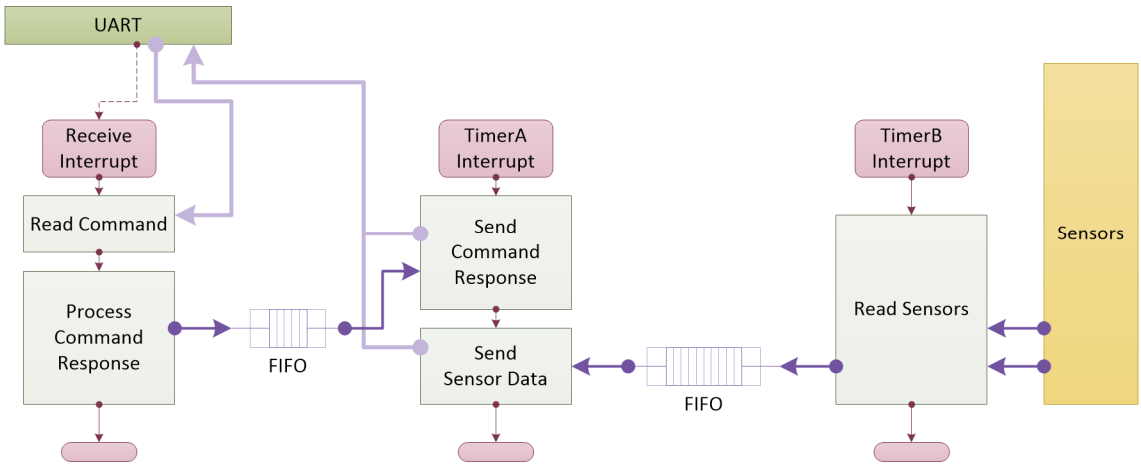


Fig.9.3 Interrupt driven firmware architecture integrated with UbiITS framework

9.1.4. Real-time task scheduling

Two interrupts were generated from independent timers while the other was generated from the MCU's UART peripheral upon receiving data packets from the remote host. These interrupts were prioritised based on the real-time task priority requirements. Reading measurements from all the sensors in a precise interval is the topmost priority of this device. Therefore *TimerB* interrupts were set to the highest priority level since all the sensors were read in the *TimerB* Interrupt Service Routine (ISR). Receiving and processing commands were set to the next level of priority since it is vital to service the requests from the remote host with minimum time delays. Receiving and processing commands in a timely manner is important whereas sending response to the commands can be delayed until the next available time schedule. For instance, regarding the sample count information requested by the host, the sending response can be delayed given that the command has been processed with a

minimal delay, i.e. sample count has been logged. Consequently, *Receive* Interrupts were prioritised higher than *TimerA* interrupts where the response to commands and sensor data (periodic sample measurements from the sensors) were dispatched to the host.

Data samples and command responses are despatched at a lower frequency than the sensor sampling rate. This implies that samples are queued in the FIFO buffer by *TimerB* ISR at every sample measurement interval and dispatched by the *TimerA* ISR in batches. While sampling intervals are strictly maintained and not interrupted by any task, dispatch intervals are relaxed and the dispatch task yields time to higher priority tasks. The latency of samples being delivered to the host can be reduced by the increasing the dispatch frequency. However this increases the active time of the communication channel and hence increases computational load on the host. On the other hand decreasing the dispatch frequency requires an increased FIFO buffer length. The sampling rate and dispatch rate can be altered by setting corresponding parameters in the firmware. These parameters can be altered such that it satisfies the application's requirements of real-time response and computational loads.

In summary the task scheduling policy was designed to guarantee the following characteristics:

1. The measurement of samples from the sensors are captured at fixed frequently with strict timing

2. Commands are received and processed with minimum delay, this can be interrupted only for sample measurements

3. Data samples and command responses are dispatched at a lower priority, yielding time for higher priority tasks.

The following priorities were set in the firmware for concurrent tasks and an illustrative time slicing diagram is shown in Fig.9.4.

$$Reading\ Sensors > Receiving/Processing\ Commands > Sending\ Command\ Response\ and\ Data$$

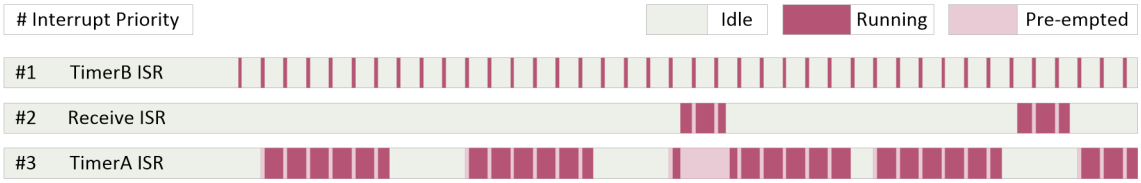


Fig.9.4 Illustrative timing diagram of concurrent tasks. Interrupt prioritises #1>#2>#3

9.1.5. Mutual exclusiveness – safe data access

The task scheduling also guarantees the mutual exclusion of critical data and peripherals, i.e. sensors are accessed only by the *TimerB* ISR, all the data including the command responses and sensor measurements are sent through the UART peripheral by *TimerA* ISR and the data is read from the UART peripheral in the *Receive* ISR. Two FIFO buffers are used to exchange sample measurements and command responses across concurrent tasks. Asynchronous FIFOs with no copy operations, defined in the UbiITS framework (section 4.3.6) were basically used for these asynchronous concurrent task operations.

9.1.6. Remote host

It was also necessary to share the single UART connection in this device for sending periodic sample measurements and from the sensors and the command responses. Generic data transport message service defined in the framework (section 4.4.2) was used to encode both samples and command responses. On the other side of the sensor device, a PC-software based with generic serial module is used to decode the data and interconnect the device to the rest of the system.

9.2. Custom calibration device

The firmware of previously built sensing device (Section 9.1) has been reprogrammed to function as a calibration device that assists measuring latencies in other proprietary devices. The process used here to automatically measuring latencies is exactly as was defined in the framework and detailed in section 4.6.4.

9.2.1. Firmware architecture and latency measurements

Extra IO ports available in the MCU were used to generate synthetic patterns in the proprietary device being calibrated. For example, latency in a video channel from an optical device/video camera can be estimated by connecting a light emitting diode to the IO ports and the latency in the audio channel of an audio capture device can be estimated by connecting a buzzer to the IO ports. Fig.9.5 shows the firmware architecture used in the custom calibration device. The predominant reason for building such a device on an embedded platform is that the timings involved in the customised calibration device are much more predictable and controllable since its firmware is operating on bare metal. The device was designed to function as an external hub to a PC that is not affected by the non-deterministic timing characteristics of a general purpose OS.

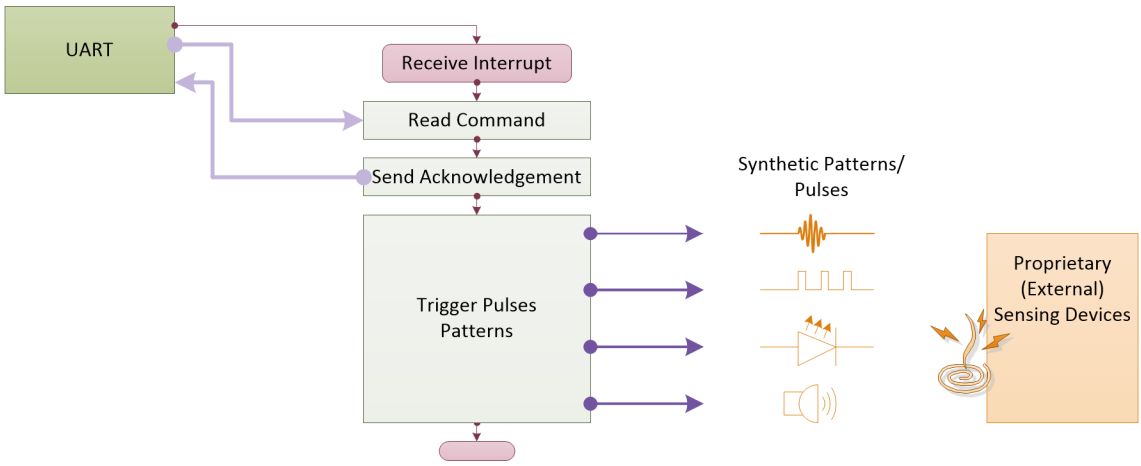


Fig.9.5 Firmware architecture for custom calibration device

9.2.2. Sample latency measurement – EEG device

In order to demonstrate the functionality of this calibration device and the process of calibration, the Nexus32 psychophysiological measurement device was chosen and the latency in its data channels measured (Fig.9.6). A PC with Windows 7 32bit platform was used as the central system and both the calibration device and psychophysiological measurement device were connected to the central system. Basically the purpose of the setup was to measure the latency of psychophysiological signals with reference to the central system. The external circuitry was designed such that a synthetic pattern generated by the calibration device was fed directly into the psychophysiological measurement device's digital input channels while maintaining the voltage levels of the pulses within the input voltage limits and reference levels. The pulse signals generated were also connected to an oscilloscope so that they could be monitored independently (Fig.9.7).

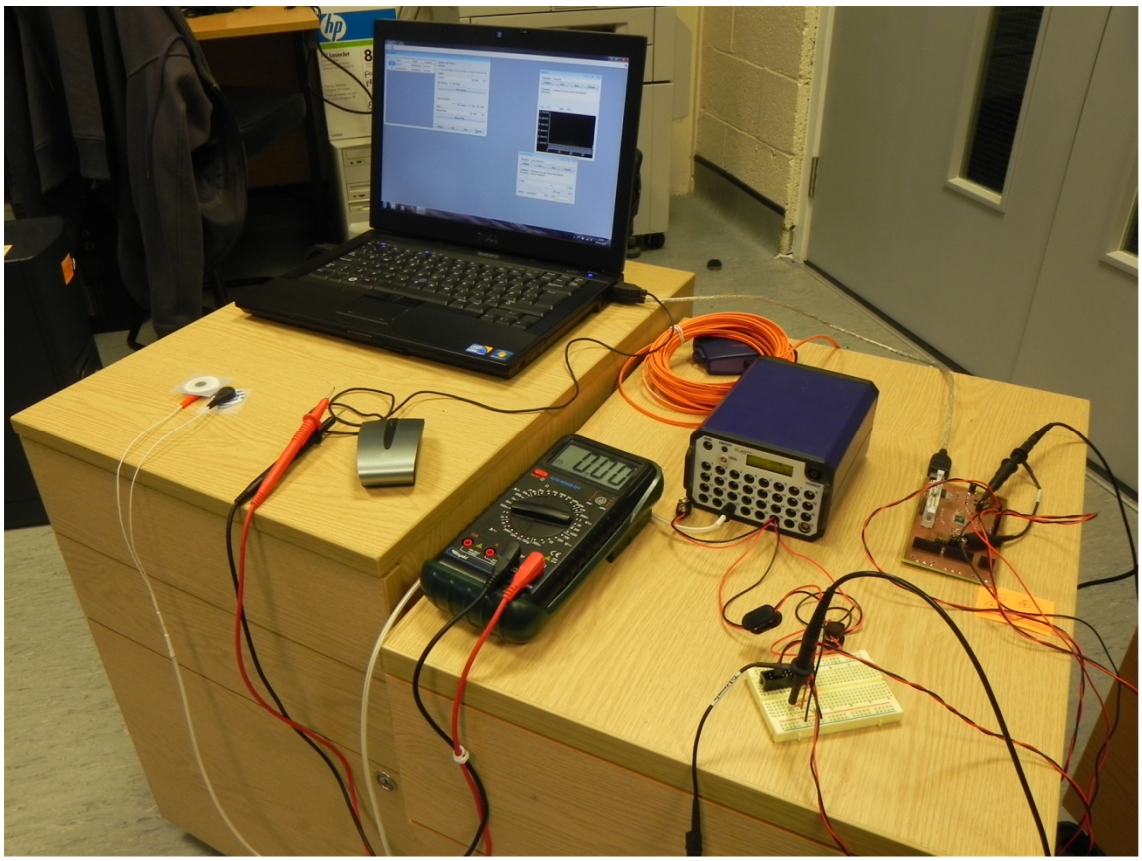


Fig.9.6 Test rig for measuring latency in psychophysiological measurement device's channels

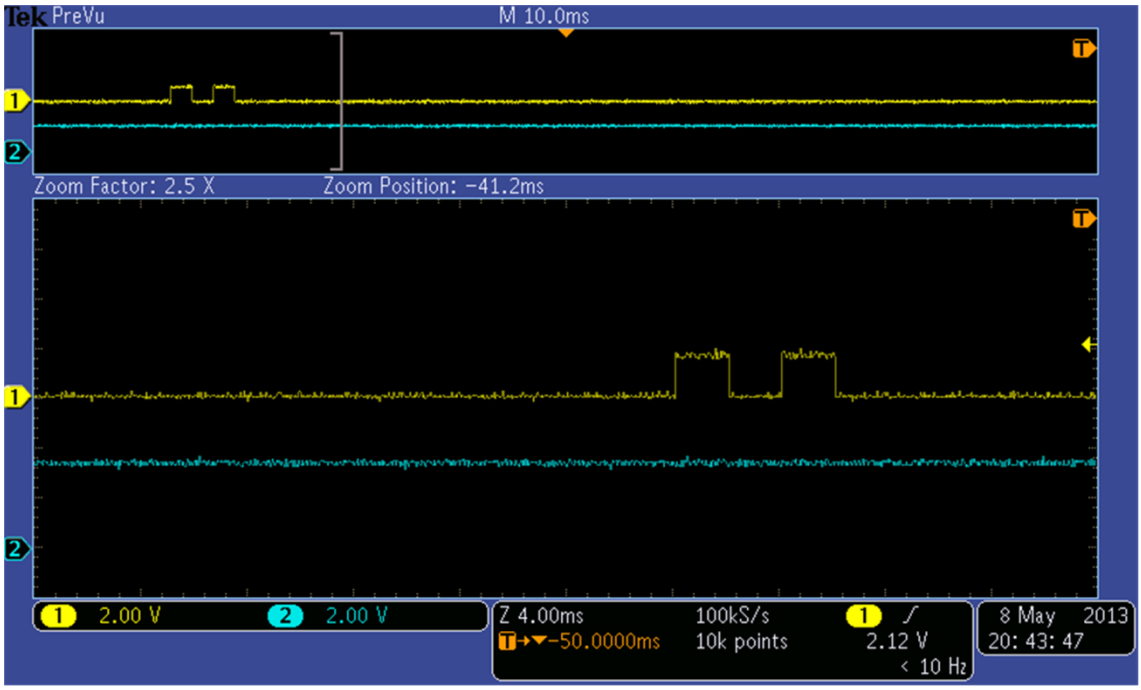


Fig.9.7 Digital pulses induced in the signal captured by an oscilloscope

9.2.3. Automatic latency measurement module

A dedicated calibration module was designed in the central system to perform calibration tasks, connect and operate as host to the calibration device. The sequence of tasks performed

was derived from the calibration procedure defined in the framework (section 4.6.4) and listed as below:

1. Commands are sent to the calibration device by the calibration module to create synchronisation pulses and simultaneously synchronisation event is raised in the system.

2. Instantaneous sample counts of all data streams in the systems and a timestamp are recorded by the global event router when the synchronisation event is raised.

3. Once the command is received by the calibration device, an acknowledgement is sent and discrete patterns triggered.

4. Another event is triggered by the synchronisation module upon receiving the acknowledgement sent by the device.

5. Another timestamp is recorded when the acknowledgement event was triggered, sample counts are not required at this time.

6. Steps 1 to 5 are repeated for numerous times to find an average estimate of the latency over many trials.

The sequence of actions and the latencies involved are shown with the physically connected hardware devices in Fig.9.8; this is a practical analogy of the concept illustrated in Fig.4.15.

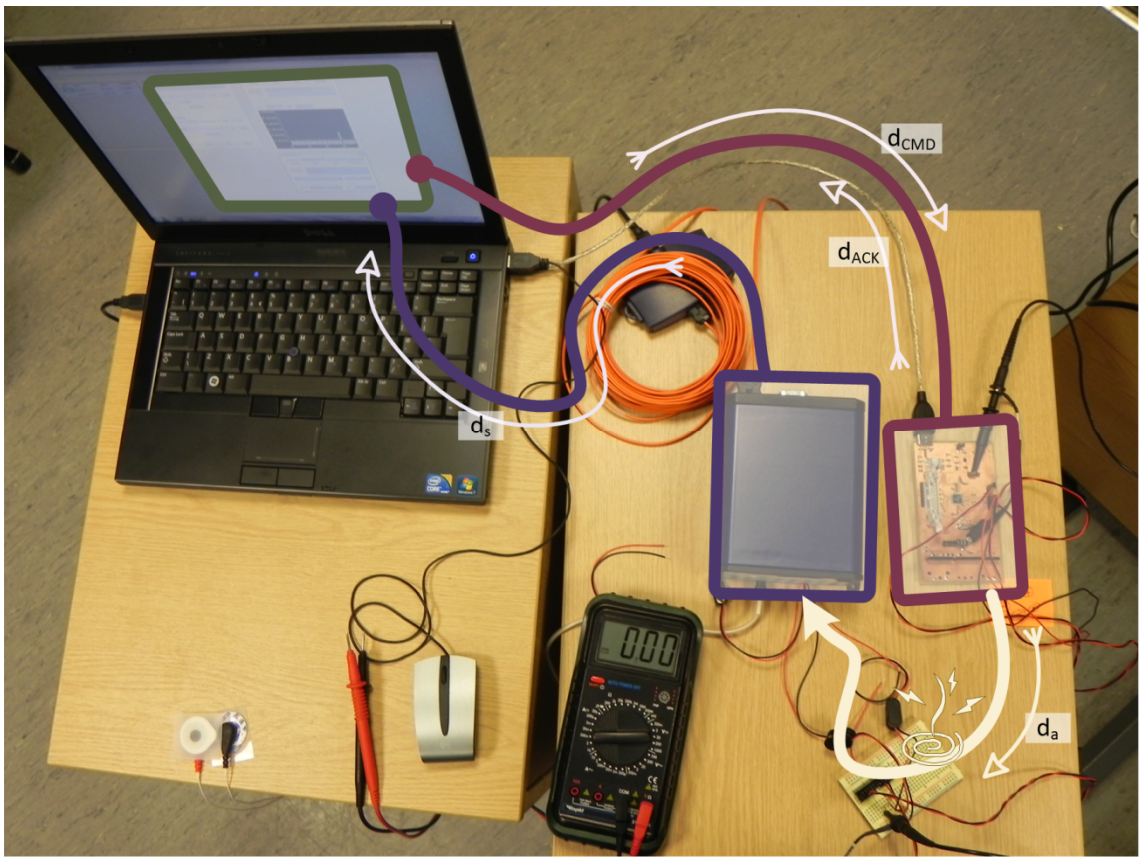


Fig.9.8 Physically connected devices with latencies involved in the calibration process, this illustration is a practical analogy of the concept illustrated in Fig.4.15

9.2.4. Latency estimation over multiple trials

The time taken to trigger synchronisation patterns upon receiving the command from the host can be assumed to be negligible, since it is actually the time taken to pump electrical pulses through a copper wire. Consequently, the actuation latency of a synchronisation pattern only involves the command latency. Latencies of pulses accumulated over multiple trials are shown in Fig.9.9. The time of the synchronisation command sent was indicated as time zero, i.e. timestamp of initial synchronisation event in step2. Red bars indicate the latencies of the pulses from the associated synchronisation pulses triggered by the device until they have been received by the system. In some instances acknowledgements were received after the signals were received by the system. These were due to general purpose OS tasks scheduling issues and are considered as outliers as indicated by the blue bars. The average command latency was indicated as zero in many instances although it is a nonzero value. The main reason for this was that the timer API used to acquire timestamps in the system was limited to millisecond timestamps. The USB/UART peripheral was enabled with RTS-CTS hardware flow control which therefore provides very low latencies. Nevertheless this did not significantly affect estimates since the signal latencies are of a higher order when compared to the

command latencies. The distribution of signal and command latencies accumulated over multiple trials is shown in Fig.9.10.

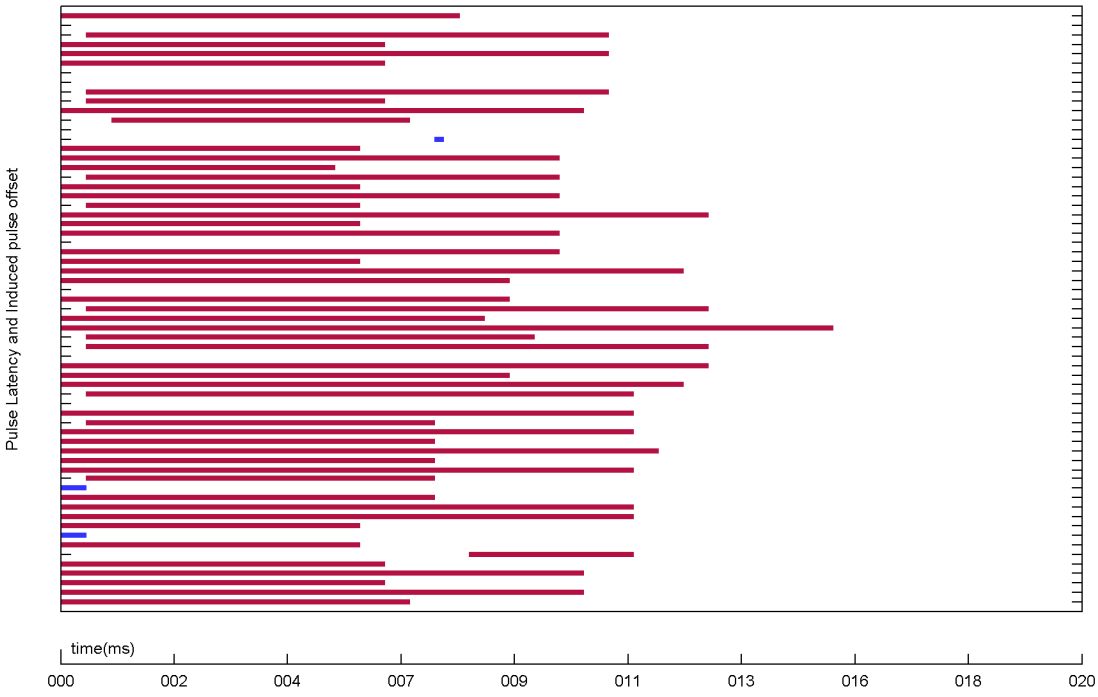


Fig.9.9 Results accumulated over multiple synchronisation trial. All trials are temporally aligned to the synchronisation command sent to the device. Red bars indicate the latencies, from the pulses triggered by the device until they received by the system. Blue bars indicate outliers - where the acknowledgements are received by the system later than the signal

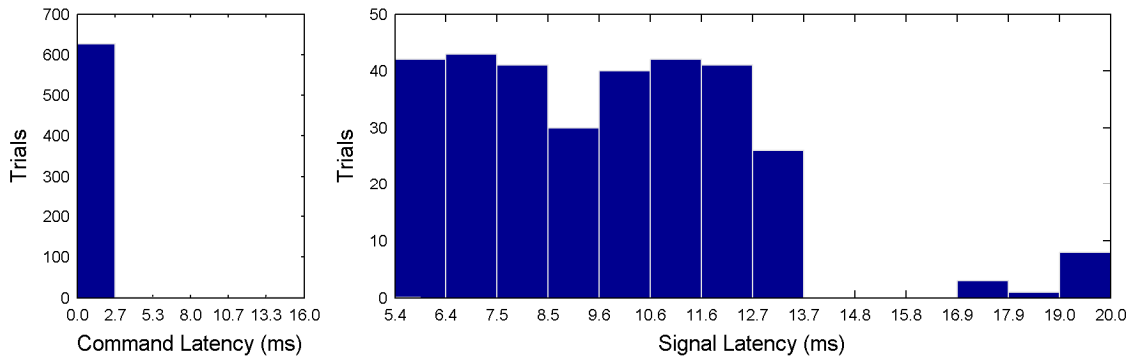


Fig.9.10 Distribution of signal and command latencies accumulated over multiple trials

9.3. Summary of hardware implementation

This chapter has demonstrated how the UbiITS framework can be implemented directly onto hardware by providing an exemplar sensor design. The firmware architecture described can be used as reference architecture for designing devices with plug-and-play support, to be used in a system based on the UbiITS framework. Subsequently this chapter showed how a hardware

device can be designed to measure latencies in preparatory sensing devices based on the automatic latency measurement procedure defined in the framework. These hardware implementations principally demonstrate the framework's capability to handle hardware design and integration problems.

Chapter 10. Framework metrics and discussion

The case studies provided in this thesis have highlighted the novel outcomes delivered by applying the UbiITS framework and substantiated why such a framework is needed. They have successfully demonstrated the outcomes of employing the framework from the individual application domain's point of view. Subsequently a model for implementing framework components directly on the hardware was also discussed. Previous chapters presenting these case studies successfully addressed the thesis objectives of refining, verifying and evaluating the framework.

This section provides a summary evaluation of the UbiITS framework and its conceptual approach from the framework's overall point of view as a generalised summary. The information and numeric metrics given here are intended to be used as a reference blueprint for future use case implementations of the framework. Nevertheless the whys and wherefores of various components and their functional capabilities have been shown mainly via the case studies' practical examples.

10.1. Evolution of the framework in parallel with case studies

While the case studies are concrete examples of the UbiITS framework implementations, they were also used to refine the evolving framework as stated in Chapter 3. It is crucial to recognize that a framework evolves in parallel with the multiple case study applications. It is generally not feasible to define an effective framework merely based on a single application. This is because the ability to outline generalised concepts and abstractions for multiple scenarios can only be achieved by physically building the application systems and discovering the abstractions and component reuse cases across multiple applications [146]. Consequently, the evolution of UbiITS framework and the development of applications occurred in parallel, aligning with the framework evolution methodology (Fig.10.1). It is also not realistic to report the evolution in an incremental development manner based on each and every application scenario. It is difficult to differentiate intermediary stages of a framework that is being actively evolved, although clear distinctions can be specified across major version increments. All refinements made during the evolution of the UbiITS framework were immediately reflected in all the case study application implementations in accordance with this parallel model. Thus, it required constant rebuilding the case study application systems (i.e. recompiling and rebuilding the software) but ensured that all of them were incorporated the up-to-date framework concepts and generalisations. The UbiITS framework reported in this thesis (Chapter 4) is sufficiently justified and validated with these case studies and therefore intended to be the initial major release at this stage.

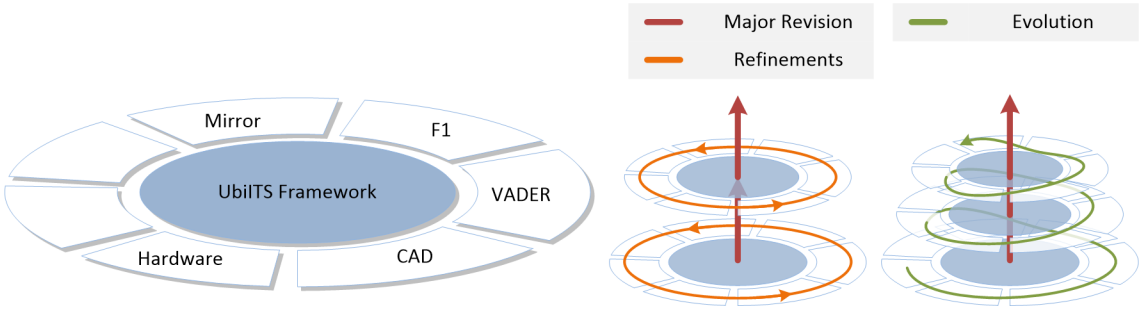


Fig.10.1 The conceptual model used for the evolution of UbiITS framework including major version increments and iterative refinements and case studies. (see: Fig.3.2) This illustration also indicates the future evolution cycles.

10.2. Software implementation metrics from the case study applications

Identifying the requirements of every new application is a challenge in building a framework. The main reason for this is the difficulty in foreseeing all the different ways in which custom implementations will want to use the framework as a solution in their domain. Although the usefulness of a framework can be subjectively claimed and debated, principally it is difficult to define the usefulness [267]. Consequently, actual functional demonstrations provided via case studies plays a vital role on defining the UbiITS framework's usefulness. As an addition to the functional characteristics revealed through the case studies, the framework's software-based metrics were produced from the application implementations. The supportive results established here can be used by future implementers to understand and interpret the framework's software-based aspects and further assist them in deciding to choose UbiITS.

UbiITS was predominantly defined to be generic and independent from any one specific application scenario. However, the usefulness of a framework can only be realised when it is being physically implemented. The software implementations associated with the case studies are only frontends while the core of the framework resides separately as a generic piece of work. However analysing the software codes within the various application implementations can generate legitimate and supportive metrics reflecting framework's usefulness, as long as one does not try to conceive the entire UbiITS framework merely based on the software code itself.

10.2.1. Implementation platforms and portability

Case study application implementations including raw framework codes were designed to follow object oriented programming principles. Table 10.1 and Table 10.2 illustrate programming languages and platforms used in the various application implementations. These implementations not only demonstrate a widespread use of multiple languages and platforms but they are also simultaneously interoperated within one system. This diversity and flexibility

is achieved by maintaining the framework concepts and components in generic and abstract forms before writing them into a language specific code. This language independent approach guarantees the adopting of the framework into various platforms, utilizing their natively available programming languages, toolsets, APIs, etc. on those platforms.

Application Implementation	C	C++	C#	C++/CLI	VB	MATLAB	Flash
CAD							
Device APIs Interface	✓	✓			✓		
Device Modules			✓	✓			
Distributed Control			✓				
GUI			✓				
Post Capture Data Analysis		✓				✓	
F1							
Device APIs Interface	✓		✓				
Device Modules			✓	✓			
Distributed Control			✓				
GUI			✓				
Post Capture Data Analysis						✓	
VADER							
Device APIs Interface	✓	✓					
Device Modules		✓					
Distributed Control		✓					
GUI		✓					
Real-time Data Visualisation		✓					
Mirror							
Device APIs Interface	✓	✓					
Device Modules		✓					✓
Distributed Control		✓					
GUI		✓					✓
Game							✓
Post Capture Data Analysis						✓	
Custom Sensor Device	✓						
Calibration Device	✓						

Table 10.1 Programming languages used in case study application implementations

Application Implementation	Windows	Linux	On bare metal
CAD	✓		
F1	✓		
VADER		✓	
Mirror	✓		
Custom Sensor Device			✓
Calibration Device			✓

Table 10.2 Platforms used by the case study systems

There are two main code implementations used through all case studies, namely in C++ and C#. Although the core conceptual framework is the same for these two versions, the coded implantations differ based on language specific styles and structures. For example C++ supports multiple-inheritance while C# only supports single-inheritance, function pointers versus delegates, pure virtual functions versus interfaces, etc. to name but a few. The custom built IMU sensor and the calibration devices are examples of the framework's concepts embedded directly onto hardware. These implementations run directly on the "bare metal" without any host operating systems. Furthermore it is not intended to cast all aspects of the framework into software code since several parts of the framework are not meant to be software coded but are more of reference guide for a particular setup. For example, distributed deployment and synchronisation techniques are provided as guidance for how the system should be implemented and/or in explaining a procedure.

10.2.2. Framework components usability metrics

Numbers of usage instances of components defined in the UbiITS framework were utilised as a basis for establishing usability metrics. These abstract components were designed in the code in both standalone and combination manner i.e. classes and structures in the implementations were designed to have dedicated or joint functionalities. Table 10.3 shows number of instances of the generic components defined in the framework and used in each application.

UbiITS framework Components	CAD	VADER	F1	Mirror
Unified Interface	14	9	5	4
Device State	15	10	6	6
Event Data	8	4	4	2
Stream Data	8	6	3	4
Data Source	11	7	3	4
Data Sink	10	7	3	4
Asynchronous FIFO	10	8	4	6
Asynchronous Message	4	3	3	3
Asynchronous Event Handler	5	4	1	2
Automatic Latency Measurement	1	1	1	1
Sample Count	8	6	3	3
Global Timestamp	1	1	1	1
Distributed Clock Synchronisation	2	1	0	2

Table 10.3 Number of usage instances of components of UbiITS framework used in each application implementation. Detailed data of individual classes and components can be found in Appendix 2

Code metrics proposed by Fayad et al [267] and Bansiya et al [268] for object oriented application frameworks were used as a model to establish metrics on the UbiITS framework.

Fayad et al's work further suggested the appropriateness of choosing alternative metrics according to the required granularity and characteristics being evaluated [267]. Consequently the metrics selected here to evaluate the UbiITS framework's usability slightly differ although they have been derived from this work. The basic difference is that the chosen metrics reflect the core framework's structural aspects and usability while Fayad et al's metrics compare two different large frameworks' (MFC [269] and OWL [270]) usability and track their stability based on the major version increments.

Classes/Interfaces	Number of Instances				Ratio with total number of classes %			
	CAD	F1	VADER	Mirror	CAD	F1	VADER	Mirror
Total Number of Classes	55	53	29	33				
Framework Classes -Extended	20	17	6	8	36%	32%	21%	24%
Framework Classes -Used raw	21	21	4	14	38%	40%	14%	42%
User Interface/General	5	6	2	1	9%	11%	7%	3%
User Interface/Device	9	9	1	7	16%	17%	3%	21%
Application related functionality	0	0	16	3	0%	0%	55%	9%

Table 10.4 Structural composition of UbiITS components in each application implementation.

The ratios of framework related components and the other parts of the applications against the total number of classes are shown in Table 10.4 highlighting the amount of reusable framework design in each application. In addition to the components used as raw from the framework, components customised or extended (i.e. inheritance) from framework classes are structured and unified, therefore reusing the design. This design reuse points implicitly to the knowledge brought from the framework domain to each application domain. Customising and extending framework classes is one of the classical software framework characteristics, i.e. inversion control [7], which plays a vital role in interoperating with multiple, diverse tools in an integrated system.

The following indices were derived to reflect the structural characteristics of the UbiITS framework. The parameters used to create structural characteristic metrics are defined and calculated as below.

$$\text{Average Depth of Inheritance} = \frac{\sum Depth\ of\ Inheritance}{Classes\ Extended\ from\ Framework}$$

$$\text{Average Composition Ratio} = \frac{\sum \text{Component Composition}}{Framework\ Classes\ used\ as\ Raw}$$

Depth of inheritance values implies reusability of the components. The composition ratio of framework components reflects the number of components used to provide a combined functionality in a class such as a device or data module combined with multiple functionalities.

Architecture Structural Indices	CAD	VADER	F1	Mirror
Σ Number of Parents	58	44	7	20
Σ Depth of Inheritance	58	44	7	17
Σ Component Composition	97	67	38	42
Average Depth of Inheritance	2.90	2.59	1.17	2.13
Average Composition Ratio	4.62	3.19	9.50	3.00
Single Inheritances	58	56	30	21
Multiple Inheritances	0	0	0	3

Table 10.5 Usability metrics established based on a study of Fayad et al [267]

10.2.3. Cyclomatic complexity and maintainability

Sample cyclomatic complexity and maintainability analyses [271] were performed on codes for the core and extended framework classes. Metrics for these codes were calculated using Visual Studio code analysis tool with Microsoft Managed Recommended and Microsoft Native Recommended rules [272]. Microsoft Visual Studio tool-chain based projects were chosen for the analysis based on the functionality and availability of the analysis toolset; these metrics are shown in Table 10.6. Similar results can be anticipated if the analysis is performed on the other projects since the underlying core framework concepts and algorithms are identical to all projects. These results can be slightly skewed because of the analysis tool characteristics, code optimisation options, and platform specific issues, e.g. managed native transitions, inline functions, etc. However these artefacts can be ignored since the main focus of the analysis is on the core framework components and the results are only for representation.

Metric	CAD	F1
Maintainability Index	80	81
Cyclomatic Complexity	412	351
Maximum Depth of Inheritance	6	7
Lines of Code	831	753

Table 10.6 Sample cyclomatic complexity and maintainability analyses performed for Microsoft Visual Studio tool chain based projects with Visual Studio code analysis tool with Microsoft Managed Recommended and Microsoft Native Recommended Rules [272]. Detailed data of individual classes and components can be found in Appendix 2

10.2.4. Supported devices and data types and ubiquity

A list of devices, data types APIs currently supported in case study applications is compiled in Table 10.7. This only demonstrates the devices and data types as examples; however, it is important to understand that the framework components are purely generic and intended to be extended by the application implementer for future applications. This further emphasises the concept of ubiquitous-ness of the UbiITS framework, i.e. as long as devices and tools are designed to follow specified interfaces and rules, then they are guaranteed to work within an integrated system designed based on the UbiITS framework.

Modality	Device/API	Stream	Event	Data Type	Frequency	Description
CAD						
CAD Logging	Siemens NX/UG Open API		x	String	-	Event description generated by customised UI
Spread Sheet Logging	Microsoft Excel/VBA		x	String	-	Excel spread sheets events
EEG	Nexus 32/API	x		19 x float	2048Hz	19 x EEG Channels corresponding different locations on the head
GSR, EMG, Heart Rate	Nexus 32/API	x		1 x float	2048Hz	1 x Bipolar Channels each sensor
Eye Tracking 1	Tobii X50/ ClearView API	x	x	4 x long + 12 x float	50Hz	Left and right eye – gaze – X,Y, pupil diameter, distance, validity
Eye Tracking 2	Arrington Research API	x	x	10 x float + 1 integer	50Hz	Left and right eye – gaze – X,Y, pupil position, width, aspect, fixation, validity and region events
Eye Tracking 3	Tobii TX300/ ClearView API	x	x	4 x long + 12 x float	300Hz	Left and right eye – gaze – X,Y, pupil diameter, distance, validity
Key Press	Keyboard/Windows Hook		x	1 x integer	-	Key code
Mouse	Mouse/Windows Hook	x	x	2 x integer + 1 x byte + 1 x integer	-	X,Y cursor position button click, wheel
Screen Video	Intensity Shuttle/DeckLink SDK	x	x	1280×720 x 3 byte	30Hz	720p hi definition (HD) screen capture. video/snap shot
Environment Video	Sony Playstation Eye/Open CV	x		640 × 480 x 3 byte	30Hz	VGA video capture
F1						
Game Data	Codemasters Propriety UDP	x		38 x float	50 Hz	Car Telemetry data streamed via UDP connection by Codemasters game engine
Game Controller Data	Logitech Steering Wheel/Windows Raw Input	x		4 x float + 27 bits	50 Hz	4 axis + 18 Buttons + 2 sequential shifters + 7 H shifters
EEG	Nexus 32/API	x		19 x float	2048Hz	19 x EEG Channels corresponding different locations on the head
Motion Sensing	Custom Built Devices/USART/COM	x		3 x 3 x 2 byte	50Hz	Gyro + Accelarometer + Magnetometer, triple axis
Audio Mute Control	Windows Message APPCommand		x	1 x integer	-	Windows Message Command for audio volume control and mute
Observer Events	Direct Triggers		x	1 x integer	-	Direct Triggers given using the GUI
VADER						
Audio	Alsa Audio	x				
Video	Sony Playstation Eye/Open CV	x		640 × 480 x 3 byte	30Hz	VGA video capture
Screen Capture	VTK	x		1980 x 1080 x 3 byte	10 Hz	VTK window image rendering
Snapshot	VTK		x	1980 x 1080 x 3 byte		
Annotation	Software Comment Box		x	Strings and Numbers - Variable		Text Annotation with Tags
Web Annotation	Software Webserver		x	Strings and Numbers - Variable		Form HTTP/POST Text Annotation with Tags
Click Interactions	Xwindow/VTK		x	1 x integer		CAD Model View Interactions VTK interactors
3D Navigation Space Pilot	3Dconnexion API	x	x	6 x float + variable		6 DOF 3D navigation and programmable input
Mirror						
Game	Custom Built Game	x	x	2 bytes + 4 bytes	10Hz	Brain Activity Event Triggers (in) + Game Logging (out)
EEG	Thought Technology/Connection Instrument SDK	x	x	8 x 5 float + String	50Hz	8 channels of EEG Data, Max, Min, high & low Thresholds + Event Comment
Video	Sony Playstation Eye/Open CV	x		640 × 480 x 3 byte	30Hz	VGA video capture

Table 10.7 List of devices, data types, APIs supported in case study applications.

10.3. Functional characteristics

Individual components of the framework are designed for the purpose of handing technical diminutives. Every component and definition provided in UbiITS framework addresses a specific low-level technical problem that arises in constructing a DCIS. It has been shown through case studies that these primitives are assembled together as a collection to solve a higher level and complex but more generic problem. Consequently it becomes important to consider the framework's functional characteristics in higher level terms. The broader functional characteristics required by each case study is listed in Table 10.8, which evidences the suitability of the UbiITS framework to address these border-scaled inherent functional requirements.

Functional Characteristics	CAD	VADER	F1	Mirror	Hardware
Multimodal data and interfaces	✓	✓	✓	✓	
Dynamic and Reconfigurable system	✓	✓			
Long-term capture	✓	✓			
Temporal, microscopic activities	✓		✓	✓	
Distributed tools	✓	✓			✓
Collaborative Interaction		✓			
Smart and Interactive Environment		✓			
Closed loop systems			✓	✓	
Human in loop			✓	✓	
Human Computer Interaction	✓			✓	
Complex data handling	✓	✓			
Hardware deployment					✓

Table 10.8 Case studies and the key functional features required by the systems. These requirements are later addressed by the architecture

10.4. Comparison with the literature

An extensive search through the literature revealed that there are no all-inclusive frameworks or architectures available for addressing the problems which arise in building a DCIS. In essence, there are no significantly matching solutions to directly derive techniques and concepts that could have been used to carry out a progressive or comparative improvement approach. Nevertheless, the generic solution researched and developed this work has been contested against any ad-hoc or domain specific counterparts. Works from the literature were taken into account which address similar concepts or where any overlap with the presented solution is found. Table 10.9 re-examines the concepts and functionalities extracted from previous works alongside the UbiITS framework and demonstrates that it has successfully addressed the key gaps identified.

The framework comprises a number of subparts which are focused particularly towards specific commonly identified problems, e.g. synchronisation, interfacing multimodal tools, etc. Standard techniques and concepts from the literature related to those specific problems were accumulated and used to build up the fundamental principles and concepts. This basic information is often generic and therefore adapted to match the requirements of a DCIS and then used throughout the framework development. Therefore partial information related to the parts of the solution was gathered from various literature and then combined into the framework itself.

In addition to the rudimentary concepts extracted from the literature, and initially used identify the requirements and subsequently primed the development of the framework, every selected case study has also informed various problems that could potentially arise in a DCIS. These inputs were also combined with the initial requirements in order reform and refine the framework's fundamental concepts, definitions and components.

	Temporal Synchronisation	Events	Streams	Real-time Data Accessibility	Real-time Interaction with environment	Data Transport	Unified interface	Device Status/Failure Management	Commodity Hardware /Software, OS	Ubiquitous , Runtime flexibility, reconfiguration	Tool Kit/Modular Building	Graphical development	Generic /Abstract Definitions	Reference Architecture	Trans-domain applicability -Framework	Target Application
Social Signal Interpretation (SSI) Framework [121], [122]	●		●	●							●					Online Emotion Recognition
Open Interface (OI) Framework [22], [123]	●		●		●							●				Multimodal Input Interactions
EyesWeb Extended multimodal interaction (EyesWeb XMI) Architecture [124], [125], [114]	●		●						●			●				Multimodal data mapping on to Audio visuals
Pure Data [128]–[130]			●	●							●	●			●	real-time multimedia content, audio visuals
User-centred Experiment and Logging Framework for Interactive Information Retrieval [34]	●								●					●		behavioural data logging in PC tasks
Tracking Real-Time User Experience (TRUE) Architecture by Microsoft Game Studios [131]	●	●														Game data logging
Multimodal Data Capture and Analysis of Interaction in Immersive Collaborative Virtual Environments [132]									●					●		Multimodal VR interactions capture
VR-Juggler [110], [133], [134]							●	●		●	●		●			VR
VRPN [79]						●	●						●			VR
Automatic Event-based Synchronization of Multimodal Data Streams from Wearable and Ambient Sensors [93]	●		●													Physiological data capture
NIST Data Flow System [20], [96]	●		●	●		●			●							Audio, Visual
Synchronous data collection from diverse hardware [53]	●															Driving activity capture
Bio signals Studio [135]			●	●												Physiological data logging + voice, video
Service-oriented smart home applications: composition, code generation, deployment, and execution [136]								●								Smart user spaces
A Database-Oriented Wrapper for Ubiquitous Data Acquisition/Access Environments [54]							●	●		●						Video, Identification
Finite-State Machine Based Distributed Framework DATA for Intelligent Ambience Systems [137]		●						●								Intelligent ubiquitous systems
W3C - Multimodal Architecture and Interfaces [138]		●			●			●					●		●	User interface
Ubiquitous Synchronisation and Interaction Architecture	●	●	●	●	●	●	●	●	●	●	●		●	●	●	General purpose DCIS

Table 10.9 Revisiting the Literature: Comparison of concepts and functionalities from the literature alongside UbiITS Framework

10.4.2. Functional novelties

In general none of the previous works given in Table 10.9 considered the in-depth technical details to the level and granularity that the UbiITS framework has considered. Moreover, categorical definitions of techniques and components which aid the solving of fundamental technical issues, i.e. hardware implementation, deployment configurations, task concurrency, multi-threading and platform independency, are also novelties included in the UbiITS framework. These are important features to a DCIS but overlooked in previous research. The majority of previous work could have certainly encountered the same technical issues but did not consider addressing them in a holistic manner, or may have addressed them in a superficial manner. The generic definitions included in the framework enable it to work closely with, and provide fine grained control over the fundamental technical problems. It is presumed that it is appropriate to make the users aware of these issues rather than ignoring or blurring them behind assumptions. Consequently the UbiITS framework brings forward these low-level issues, allowing the system implementers to use standard components supplied with the framework, extend/customise the components to address the issues, or else discount them in some cases where they have trivial effect on the application.

Dissimilar solutions are constrained to a single time synchronisation technique, UbiITS offers multiple techniques allowing the choice of one that is most suited for the purpose. The novel intra-stream synchronisation approach provided enables multiple, multimodal data streams to be synchronised. This technique is originally derived by combining source data based approaches [93], [106] and reference broadcast synchronisation based approaches [155]. In order to support multiple and mixed modalities it uses a reference calibration device. This method is free from the internal clocking mechanism of each device/component, can be applied to any type of data stream and is independent from the modalities. This method includes the overall, effective latency of the physical action beyond the latencies which occur inside the associated software systems.

10.4.3. Exportability and generic framework

A main novelty of this work is that no other work has proposed a framework as a generic solution to address the problem of constructing a DCIS. Providing a solution in the form of a framework approaches the problem in a highly generic fashion. Conversely, previous works from the literature have mostly focused on solving a part of a problem within a particular domain, and consequently, they become ad-hoc solutions constrained to a those domains or environments. Although constraining the problem to a specific scenario is indeed worthwhile

in delivering a specialised solution, the degree of reusability of the outcome when ported to a completely new scenario is highly questionable.

An application designed to solve a specific problem scenario identifying itself as a 'framework' is also highly questionable, regardless of its success in serving the particular circumstance. Although the term 'framework' has been stretched and used with extensive interpretations in previous works [34], [135], [225], [273], [274], it has been carefully chosen and deliberately used in this thesis. Well established concepts and characteristics derived from software-framework development methodologies were incorporated into this solution and its process of evolution; for example, generalisation, embodiment of domain knowledge and design of components and generative architecture for maximised reuse, to name a few [7], [9], [144], [149]. Consequently, these features inherently embedded in the UbiITS framework provide a systematic justification for it to be called a 'framework'. Similarly the term multimodal is often exploited to represent 'more than one' modality, rather than being open to 'any' modality [20], [275], [276]. However, multimodal tools in the UbiITS framework are conceived as multiples and diversities of tools; essentially enforcing the openness to 'any type' of current and future tools, therefore retaining by default its ubiquitous nature.

For instance, SSI framework [122] and OI Interface framework [22] are two specialised solutions for specific purposes, favourably supporting a wide range of multimodal devices. Although the OI framework has indubitably recognised the need of generic components, both of these solutions lack the definitions of clear abstractions so that diverse and future tools and data types can be effortlessly supported. Various tailored modules are added by the developers to enrich the support across a diversity of tools. A tendency to add various modules and absorb them within the solution is apparent in this work [22], [122]. This approach leads to an extensively inflated solution rather than a generic, compact and reusable solution. In contrast UbiITS depends on internally defined generalised abstract modules, which can be extended/customised by the users. An unambiguous separation between core components and extended components is drawn here. In this manner customised components will stay external to the framework, whereas the core of the framework is compact and abstract therefore increasing the reusability potential of the internal design. Following the published material on UbiITS [277], [278] the SSI framework appears to have adopted this concept of abstraction in some of its basic components, i.e. events and streams [279] in a recent version (v0.92) which was not included in earlier versions (v0.8).

Instead of developing an ad-hoc solution and later trying to export it to other scenarios as a "that is all you need to have" solution, this work predominantly abstracts the problem from initial conception, and later walks system developers through the process to find their own

bespoke implementations. This novel approach to DCIS development in return offers a highly exportable solution to various domains and new scenarios. The fundamental concept of abstraction and generalisation in framework development theory is particularly intended to give plenty of room for extensions and reusability while maintaining a stable architecture [7], [147]. Essentially, the framework solution provided in this work functions as a modular toolkit with generic and extendable components that help to build domain specific solutions in widespread disciplines.

Chapter 11. Conclusions and future directions

The Ubiquitous Integration and Temporal Synchronisation (UbiITS) Framework is a generic solution to address the technical issues in building a Ubiquitous multimodal DCIS. This framework provides a structure and components to build a system that links the physical interaction space to the data cyber space in real-time. This thesis has presented the basic definition of the framework and supporting evidence to justify its conceptual, functional and methodological correctness. UbiITS has addressed the knowledge and technological gaps identified from the problem domain in a generic manner so that maximising its exportability and exploitability across a wide range of application scenarios. Generic tools and techniques provided in the framework enhance design reuse and inherently demonstrate the potential to standardise systems design for DCIS.

Four different case study applications employing UbiITS framework were provided to substantiate why such a framework is necessary, evidence its capabilities to handle application scenarios and outcomes all of which were achieved by employing the framework. While these case studies are used to substantiate and justify the framework, they are also novel systems within their own domain. These case studies were real-life problems which could not have been solved without the UbiITS framework, however once the technical requirements were addressed and foundation architecture was established by the framework, then these case studies opened up new lines of research.

Beyond serving the internal lab-based application scenarios, the framework has been deployed in other external fields, for example real-time neurofeedback applications neuroscience research. This essentially shows the framework's maturity level and state of readiness for public distribution. Recognising the importance and successful use of the framework from the internal and external application scenarios, successively a patent application procedure has been initiated and is currently in progress.

The original hypothesis has been provided below:

Building a framework for integrating multimodal data capture, interaction tools and time synchronisation will standardise system design and therefore enhance design reuse and interoperability.

This thesis has successfully met various research objectives and the hypothesis has been justified to be valid as a consequence of the evidence provided.

11.1. Future refinements and open ended evolution

The success of any framework depends on how often and to what extent it is being used in applications. Generally predicting and addressing the needs of all future applications is a difficult task. This thesis has demonstrated the success and potential of UbiITS framework at its current stage with multiple case study applications. However when it is increasingly being employed in future applications, particular needs will become transparent and will support further refinement the framework. The framework will continue to evolve as it is being continually refined; therefore the evolution of the UbiITS framework is open ended.

Higher-level architectures based on UbiITS framework

Example systems utilising the UbiITS framework were demonstrated via case studies. The system architectures presented in these case studies demonstrate their individual potential within their domain. These case study applications are higher-level systems operating on top of UbiITS. They also exhibit the potential to evolve as separate higher-level domain specific frameworks or reusable architectures. Similar to the case studies, there are opportunities to develop domain specific, specialised and higher-level frameworks or system architectures. These higher-level structures will have the freedom to target on a narrower and concentrated problem. There is no need to work on the low-level technicalities since they are already addressed by UbiITS. Basically, these higher-level structures will inherit all the indispensable capabilities of from the framework and possibly build their architectures on the foundation or skeleton provided.

Ubiquitous and cyber physical systems

The scope for the UbiITS framework had been selected as ‘data capture and interaction’; this had been deliberately chosen to encourage wider usage. The problem domain addressed by UbiITS is one of the core problems in many disciplines, such as cyber physical systems, smart environments, human-in-loop systems, perpetual/life-long recording, knowledge capture systems, neurocybernetics, human machine interfaces, etc. All of these applications share the common need of ubiquitous, multimodal data capture and interaction. Including, but not limited to, the above mentioned disciplines, apparently have the potentials to adopt this framework. UbiITS framework has laid out the ground for its abstractions at various levels as to be adequately generic and to enhance reusability. It is logical to expect higher level applications adopting the framework may well redefine the abstractions to align with their needs.

Public distribution issues

The framework's core concept, functionality and systematic correctness has been tested and stabilised adequately at this present stage. However, further improvements may be required on the software code before it can be distributed to the public or a third party. These improvements are mostly related to the format of the distributions and intellectual property related issues, for example distributing open access code libraries, binary executable, abstract definitions or directly deploying the framework for a client. In addition to the currently existing C++ and C# versions of the code, distributing the code in other programming languages may need to be considered.

Complex data systems

Scanning through the impending needs of case study applications reveals that there is a need to handle the complex multimodal data beyond capturing. At present the framework only addresses the real-time handling of the data and not post capture data management issues. Consequently a possible necessity to extend the framework's scope to include database management modules is recognized.

Cloud based systems

The current framework architecture is designed to operate on a PC or embedded platforms. This is mainly based on the need for dedicated and capable platforms that can handle time critical, high bandwidth and computationally demanding tasks. An increasing tendency to move personal computing towards cloud-based systems emphasises the need to consider cloud compatible architecture for all kinds of applications. Building a cloud-based architecture for the UbiITS framework can provide potential advantages in a number of areas; for instance, distributed deployment of devices and parallel synchronous environments, etc. Although cloud-based back end platform and a standard PC-based system possess various similarities, the integration of core architecture running in the cloud backend with devices connecting to the cloud frontend may require additional components within the framework.

References

[1] J. Elson and D. Estrin, “Sensor networks: A bridge to the physical world,” *Wirel. Sens. Netw.*, pp. 3–20, 2004.

[2] G. Buscher, J. Gwizdka, J. Teevan, N. Belkin, R. Bierig, L. Elst, and J. Jose, “Preface: Proceedings of the SIGIR 2009 Workshop on Understanding the User – Logging and interpreting user interactions in information search and retrieval,” *Proc. Workshop Underst. User - Logging Interpret. User Interact. Inf. Search Retr. UIIR- 2009*, pp. i–v, 2009.

[3] R. Baheti and H. Gill, “Cyber-physical systems,” *Impact Control Technol.*, pp. 161–166, 2011.

[4] E. A. Lee, “Cyber Physical Systems: Design Challenges,” in *2008 11th IEEE International Symposium on Object Oriented Real-Time Distributed Computing (ISORC)*, 2008, pp. 363–369.

[5] J. D. Hollan and E. L. Hutchins, “Opportunities and challenges for augmented environments: A distributed cognition perspective,” *Des. User Friendly Augment. Work Environ.*, pp. 237–259, 2010.

[6] R. E. Johnson and B. Foote, “Designing reusable classes,” *J. Object-Oriented Program.*, vol. 1, no. 2, pp. 22–35, 1988.

[7] M. Mattsson, M. Mattsson, and M. Mattsson, “Object-Oriented Frameworks - A survey of methodological issues,” 1996.

[8] R. E. Johnson, “Frameworks = (components + patterns),” *Commun ACM*, vol. 40, no. 10, pp. 39–42, Oct. 1997.

[9] M. Fayad and D. C. Schmidt, “Object-oriented application frameworks,” *Commun ACM*, vol. 40, no. 10, pp. 32–38, Oct. 1997.

[10] P. Kruchten, “Mommy, where do software architectures come from,” in *1st International Workshop on Architectures for Software Systems, Seattle, WA*, 1995.

[11] M. Fowler, *Patterns of enterprise application architecture*. Boston: Addison-Wesley, 2003.

[12] L. Bass, P. Clements, and R. Kazman, *Software architecture in practice*. Upper Saddle River, NJ: Addison-Wesley, 2013.

[13] Z. Yu, M. Ozeki, Y. Fujii, and Y. Nakamura, “Towards smart meeting: enabling technologies and a real-world application,” in *Proceedings of the 9th international conference on Multimodal interfaces*, 2007, pp. 86–93.

[14] S. Oviatt, A. DeAngeli, and K. Kuhn, “Integration and synchronization of input modes during multimodal human-computer interaction,” in *Referring Phenomena in a Multimedia Context and their Computational Treatment*, 1997, pp. 1–13.

[15] B. MacIntyre, E. D. Mynatt, S. Voida, K. M. Hansen, J. Tullio, and G. M. Corso, “Support for multitasking and background awareness using interactive peripheral displays,” in *Proceedings of the 14th annual ACM symposium on User interface software and technology*, 2001, pp. 41–50.

[16] R. Cutler, Y. Rui, A. Gupta, J. J. Cadiz, I. Tashev, L. He, A. Colburn, Z. Zhang, Z. Liu, and S. Silverberg, “Distributed meetings: A meeting capture and broadcasting system,” in *Proceedings of the tenth ACM international conference on Multimedia*, 2002, pp. 503–512.

[17] P. Szolovits, Y. Zhang, and others, “Real-time analysis of physiological data and development of alarm algorithms for patient monitoring in the intensive care unit,” Massachusetts Institute of Technology, 2003.

[18] A. R. Doherty, A. F. Smeaton, K. Lee, and D. P. W. Ellis, “Multimodal segmentation of lifelog data,” in *Large Scale Semantic Access to Content (Text, Image, Video, and Sound)*, Paris, France, France, 2007, pp. 21–38.

[19] C. Efstratiou, N. Davies, G. Kortuem, J. Finney, R. Hooper, and M. Lowton, “Experiences of designing and deploying intelligent sensor nodes to monitor hand-arm vibrations in the field,” in *Proceedings of the 5th international conference on Mobile systems, applications and services*, 2007, pp. 127–138.

[20] L. Diduch, A. Fillinger, I. Hamchi, M. Hoarau, and V. Stanford, “Synchronization of data streams in distributed realtime multimodal signal processing environments using commodity hardware,” in *Multimedia and Expo, 2008 IEEE International Conference on*, 2008, pp. 1145–1148.

[21] M. Ozeki, S. Maeda, K. Obata, and Y. Nakamura, “Virtual assistant: an artificial agent for enhancing content acquisition: how ambient media elicit information from humans,” in *Proceedings of the 1st ACM international workshop on Semantic ambient media experiences*, New York, NY, USA, 2008, pp. 75–82.

[22] M. Serrano, L. Nigay, J.-Y. L. Lawson, A. Ramsay, R. Murray-Smith, and S. Denef, “The openinterface framework: a tool for multimodal interaction.,” in *CHI’08 extended abstracts on Human factors in computing systems*, 2008, pp. 3501–3506.

[23] J. F. Lichtenauer, J. Shen, M. F. Valstar, and M. Pantic, “Cost-effective solution to synchronised audio-visual data capture using multiple sensors,” in *Proc. IEEE Int’l Conf’Advanced Video and Signal Based Surveillance*, 2010, pp. 324–329.

[24] J. Schumm, C. Setz, M. Bächlin, M. Bächler, B. Arnrich, and G. Tröster, “Unobtrusive physiological monitoring in an airplane seat,” *Pers. Ubiquitous Comput*, vol. 14, no. 6, pp. 541–550, Sep. 2010.

[25] “CALLAS.” [Online]. Available: http://www.callas-newmedia.eu/. [Accessed: 14-Nov-2013].

[26] “METABO.” [Online]. Available: http://www.metabo-eu.org/. [Accessed: 14-Nov-2013].

[27] “IRIS.” [Online]. Available: http://iris.scm.tees.ac.uk/. [Accessed: 14-Nov-2013].

[28] D. Davcev, V. Trajkovic, S. Kalajdziski, and S. Celakoski, “Augmented reality environment for dance learning,” in *International Conference on Information Technology: Research and Education, 2003. Proceedings. ITRE2003*, 2003, pp. 189–193.

[29] S. Kousidis, T. Pfeiffer, Z. Malisz, P. Wagner, and D. Schlangen, “Evaluating a minimally invasive laboratory architecture for recording multimodal conversational data.,” in *Proceedings of the Interdisciplinary Workshop on Feedback Behaviors in Dialog, INTERSPEECH2012*, 2012, pp. 39–42.

[30] J. Hochenbaum and A. Kapur, “Nuance: A Software Tool for Capturing Synchronous Data Streams From Multimodal Musical Systems,” in *Proceeding of the International Computer Music Conference*, Ljubljana, Slovenia, 2012.

[31] S. Stumpf, X. Bao, A. Dragunov, T. G. Dietterich, J. Herlocker, K. Johnsrude, L. Li, and J. Shen, “Predicting user tasks: I know what you’re doing,” in *20th National Conference on Artificial Intelligence (AAAI-05), Workshop on Human Comprehensible Machine Learning*, 2005.

[32] R. Atterer, M. Wnuk, and A. Schmidt, “Knowing the user’s every move: user activity tracking for website usability evaluation and implicit interaction,” in *Proceedings of the 15th international conference on World Wide Web*, 2006, pp. 203–212.

[33] K. Hercegfi, O. Kiss, K. Bali, and L. Izso, “Interface: assessment of human-computer interaction by monitoring physiological and other data with a time-resolution of only a few seconds,” *ECIS 2006 Proc.*, Jan. 2006.

[34] R. Bierig, J. Gwizdka, and M. J. Cole, “A usercentered experiment and logging framework for interactive information retrieval,” in *Proceedings of the SIGIR 2009 Workshop on Understanding the User: Logging and interpreting user interactions in information search and retrieval*, 2009, pp. 8–11.

[35] K. Hercegfi, M. Pászti, S. Tóvölgyi, and L. Izsó, “Usability Evaluation by Monitoring Physiological and Other Data Simultaneously with a Time-Resolution of Only a Few Seconds,” in *Human-Computer Interaction. New Trends: 13th International Conference, HCI International 2009, San Diego, CA, USA, July 19-24, 2009, Proceedings*, Berlin, Heidelberg, 2009, pp. 59–68.

[36] S. F. Liang, C. T. Lin, R. C. Wu, Y. C. Chen, T. Y. Huang, and T. P. Jung, “Monitoring Driver’s Alertness Based on the Driving Performance Estimation and the EEG Power Spectrum Analysis,” in *Engineering in Medicine and Biology Society, 2005. IEEE-EMBS 2005. 27th Annual International Conference of the*, 2005, pp. 5738 –5741.

[37] F. Bella, “Driver behavior in car-following: a driving simulator study,” in *European Conference on Human Centred Design for Intelligent Transport Systems*.

[38] C.-T. Lin, Y.-K. Wang, and S.-A. Chen, “An EEG-Based Brain-Computer Interface for Dual Task Driving Detection,” in *Neural Information Processing*, vol. 7062, B.-L. Lu, L. Zhang, and J. Kwok, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 701–708.

[39] “PS/2 Mouse Interfacing.” [Online]. Available: http://www.computer-engineering.org/ps2mouse/. [Accessed: 24-Nov-2013].

[40] “Wii - What is Wii? at Nintendo.” [Online]. Available: http://www.nintendo.com/wii/what-is-wii/#/controls. [Accessed: 30-Mar-2012].

[41] H. Song, H. Benko, F. Guimbretiere, S. Izadi, X. Cao, and K. Hinckley, “Grips and gestures on a multi-touch pen,” in *Proceedings of the 2011 annual conference on Human factors in computing systems*, 2011, pp. 1323–1332.

[42] “Biosemi EEG ECG EMG BSPM NEURO amplifier electrodes.” [Online]. Available: http://www.biosemi.com/products.htm. [Accessed: 30-Mar-2012].

[43] “Brain Products GmbH / Products & Applications / actiCHamp.” [Online]. Available: http://www.brainproducts.com/productdetails.php?id=42. [Accessed: 30-Mar-2012].

[44] “Trackit™ Ambulatory EEG Recorder, UK.” [Online]. Available: http://www.llines.com/products-2/products/. [Accessed: 30-Mar-2012].

[45] “OptiTrack - Tracking Tools - 3D marker and object tracking motion capture software.” [Online]. Available: http://www.naturalpoint.com/optitrack/products/tracking-tools/. [Accessed: 30-Mar-2012].

[46] “Human Interface Devices Reference (Windows Drivers).” [Online]. Available: http://msdn.microsoft.com/en-us/library/windows/hardware/ff539956(v=vs.85).aspx. [Accessed: 24-Nov-2013].

[47] “Sensor API (Windows).” [Online]. Available: http://msdn.microsoft.com/en-us/library/windows/desktop/dd318953(v=vs.85).aspx. [Accessed: 24-Nov-2013].

[48] Jonathan Corbet, Alessandro Rubini, and Greg Kroah-Hartman, *Linux Device Drivers*. O’Reilly Media, 2005.

[49] “Developer SDK, Toolkit & Documentation | Kinect for Windows.” [Online]. Available: http://www.microsoft.com/en-us/kinectforwindows/develop/. [Accessed: 24-Nov-2013].

[50] “User Manuals & Guides | Tobii.com - Tobii SDK User Manual.” [Online]. Available: http://www.tobii.com/Global/Analysis/Downloads/User_Manuals_and_Guides/Tobii_SDK_UserManual.pdf. [Accessed: 24-Nov-2013].

[51] “OptiTrack - NatNet SDK - Stream motion tracking data across networks.” [Online]. Available: http://www.naturalpoint.com/optitrack/products/natnet-sdk/. [Accessed: 24-Nov-2013].

[52] “Libogc - WiiBrew.” [Online]. Available: http://wiibrew.org/wiki/Libogc. [Accessed: 27-Feb-2011].

[53] C. Goudeseune and B. Kowitz, “Synchronous data collection from diverse hardware,” in *Conférence simulation de conduite*, 2004, pp. 245–252.

[54] Y. Akahoshi, Y. Kidawara, and K. Tanaka, “A database-oriented wrapper for ubiquitous data acquisition/access environments,” in *Proceedings of the 2nd international conference on Ubiquitous information management and communication*, 2008, pp. 25–32.

[55] K. Zhang and J. U. Kang, “Real-time 4D signal processing and visualization using graphics processing unit on a regular nonlinear-k Fourier-domain OCT system,” *Opt. Express*, vol. 18, no. 11, pp. 11772–11784, May 2010.

[56] M. Weiser, “The Computer for the 21st Century,” *Sci. Am.*, vol. 265, no. 3, pp. 94–104, 1991.

[57] M. Weiser, “The world is not a desktop,” *interactions*, vol. 1, no. 1, pp. 7–8, 1994.

[58] M. Weiser, “Some Computer Science Issues in Ubiquitous Computing,” *Commun ACM*, vol. 36, no. 7, pp. 75–84, Jul. 1993.

[59] S. Poslad, *Ubiquitous Computing: Smart Devices, Environments and Interactions*, 1st ed. Wiley Publishing, 2009.

[60] C. G. Jansson, “Ubiquitous Working Environments,” in *Designing User Friendly Augmented Work Environments*, S. Lahlou, Ed. London: Springer London, 2009, pp. 191–212.

[61] T. Kindberg and A. Fox, “System software for ubiquitous computing,” *Pervasive Comput. IEEE*, vol. 1, no. 1, pp. 70–81, 2002.

[62] K. N. Truong and G. R. Hayes, “Ubiquitous Computing for Capture and Access,” *Found Trends Hum-Comput Interact*, vol. 2, no. 2, pp. 95–171, Feb. 2009.

[63] M. S. Familiar, Mart&#xed, J. Nez, F, L&#xf3, and L. Pez, “Pervasive Smart Spaces and Environments: A Service-Oriented Middleware Architecture for Wireless Ad Hoc and Sensor Networks,” *Int. J. Distrib. Sens. Netw.*, vol. 2012, Apr. 2012.

[64] U. Saif, “Architectures for ubiquitous systems,” University of Cambridge, Computer Laboratory, UCAM-CL-TR-527, Jan. 2002.

[65] O. Etzion, *Event processing in action*. Greenwich, 74° w. long.: Manning, 2011.

[66] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister, “System Architecture Directions for Networked Sensors,” in *Proceedings of the Ninth International Conference on Architectural Support for Programming Languages and Operating Systems*, New York, NY, USA, 2000, pp. 93–104.

[67] F. Dabek, N. Zeldovich, F. Kaashoek, D. Mazières, and R. Morris, “Event-driven Programming for Robust Software,” in *Proceedings of the 10th Workshop on ACM SIGOPS European Workshop*, New York, NY, USA, 2002, pp. 186–189.

[68] F. Wang, S. Liu, P. Liu, and Y. Bai, “Bridging Physical and Virtual Worlds: Complex Event Processing for RFID Data Streams,” in *Advances in Database Technology - EDBT 2006*, vol. 3896, Y. Ioannidis, M. H. Scholl, J. W. Schmidt, F. Matthes, M. Hatzopoulos, K. Boehm, A. Kemper, T. Grust, and C. Boehm, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 588–607.

[69] J. Y.-C. Wen, G. Y. Lin, T. Sung, M. Liang, G. Tsai, M. W. Feng, and C. M. Wu, “A complex event processing architecture for energy and operation management: industrial

experience report," in *Proceedings of the 5th ACM international conference on Distributed event-based system*, New York, NY, USA, 2011, pp. 313–316.

[70] X. Gu and K. Nahrstedt, "An event-driven, user-centric, QoS-aware middleware framework for ubiquitous multimedia applications," in *Proceedings of the 2001 international workshop on Multimedia middleware*, New York, NY, USA, 2001, pp. 64–67.

[71] S. Willis, "Next Generation Data Acquisition Technologies for Aging Aircraft," 2011.

[72] D. Lafourcade, "Conduct of Flight Tests and On-Board Computing for the A380," in *Meeting Proceedings RTO-MP-SCI-162*, Neuilly-sur-Seine, France, 2005, pp. 10–1 – 10–12.

[73] C. Just, A. Bierbaum, A. Baker, and C. Cruz-Neira, "Vr juggler: A framework for virtual reality development," in *2nd Immersive Projection Technology Workshop (IPT98)*, 1998.

[74] S. Peter, A. Baumann, T. Roscoe, P. Barham, and R. Isaacs, "30 seconds is not enough!: a study of operating system timer usage," *ACM SIGOPS Oper. Syst. Rev.*, vol. 42, no. 4, pp. 205–218, 2008.

[75] R. Hastings and B. Joyce, "Purify: Fast detection of memory leaks and access errors," in *In Proc. of the Winter 1992 USENIX Conference*, 1991, pp. 125–138.

[76] K. Fraser, S. Hand, R. Neugebauer, I. Pratt, A. Warfield, and M. Williamson, "Safe hardware access with the Xen virtual machine monitor," in *1st Workshop on Operating System and Architectural Support for the on demand IT InfraStructure (OASIS)*, 2004.

[77] I. A. Neag, "Supporting hardware independence in the next generation of automatic test equipment," in *IEEE Autotestcon, 2005*, 2005, pp. 248–254.

[78] I. A. Neag and S. Gal, "A unified control interface for signal sources, sensors, switches and monitors," in *AUTOTESTCON 2003. IEEE Systems Readiness Technology Conference. Proceedings*, 2003, pp. 258–264.

[79] R. M. Taylor,II, T. C. Hudson, A. Seeger, H. Weber, J. Juliano, and A. T. Helser, "VRPN: a device-independent, network-transparent VR peripheral system," in *Proceedings of the ACM symposium on Virtual reality software and technology*, New York, NY, USA, 2001, pp. 55–61.

[80] N. Kaempchen and K. Dietmayer, "Data synchronization strategies for multi-sensor fusion," in *Proceedings of the IEEE Conference on Intelligent Transportation Systems*, 2003.

[81] F. Sivrikaya and B. Yener, “Time synchronization in sensor networks: a survey,” *Netw. IEEE*, vol. 18, no. 4, pp. 45–50, 2004.

[82] B. Blankertz, S. Lemm, M. Treder, S. Haufe, and K.-R. Müller, “Single-trial analysis and classification of ERP components — A tutorial,” *NeuroImage*, vol. 56, no. 2, pp. 814–825, May 2011.

[83] J. O. Nilsson and P. Handel, “Time synchronization and temporal ordering of asynchronous sensor measurements of a multi-sensor navigation system,” in *Position Location and Navigation Symposium (PLANS), 2010 IEEE/ION*, 2010, pp. 897–902.

[84] N. Katenka, E. Levina, and G. Michailidis, “Local vote decision fusion for target detection in wireless sensor networks,” *Ann Arbor*, vol. 1001, p. 48109, 2006.

[85] P. R. Trischitta and E. L. Varma, “Jitter in digital transmission systems,” *Norwood MA Artech House 1989 244 P*, vol. 1, 1989.

[86] M. Shinagawa, Y. Akazawa, and T. Wakimoto, “Jitter analysis of high-speed sampling systems,” *IEEE J. Solid-State Circuits*, vol. 25, no. 1, pp. 220–224, Feb. 1990.

[87] A. Dubey, G. Karsai, and S. Abdelwahed, “Compensating for Timing Jitter in Computing Systems with General-Purpose Operating Systems,” 2009, pp. 55–62.

[88] K. Yu and N. C. Audsley, “A Mixed Timing System-Level Embedded Software Modelling and Simulation Approach,” 2009, pp. 193–200.

[89] F. Cristian and C. Fetzer, “Probabilistic internal clock synchronization,” in *, 13th Symposium on Reliable Distributed Systems, 1994. Proceedings*, 1994, pp. 22–31.

[90] T. K. Srikanth and S. Toueg, “Optimal Clock Synchronization,” *J ACM*, vol. 34, no. 3, pp. 626–645, Jul. 1987.

[91] D. L. Mills, “Internet Time Synchronization: the Network Time Protocol,” *IEEE Trans. Commun.*, vol. 39, pp. 1482–1493, 1991.

[92] R. Gusella and S. Zatti, “The Accuracy of the Clock Synchronization Achieved by TEMPO in Berkeley UNIX 4.3BSD,” *IEEE Trans Softw Eng*, vol. 15, no. 7, pp. 847–853, Jul. 1989.

[93] D. Bannach, O. Amft, and P. Lukowicz, “Automatic event-based synchronization of multimodal data streams from wearable and ambient sensors,” *Smart Sens. Context*, pp. 135–148, 2009.

[94] E. Biersack and W. Geyer, “Synchronized delivery and playout of distributed stored multimedia streams,” *Multimed. Syst.*, vol. 7, no. 1, pp. 70–90, 1999.

[95] F. Boronat, J. Lloret, and M. García, “Multimedia group and inter-stream synchronization techniques: A comparative study,” *Inf. Syst.*, vol. 34, no. 1, pp. 108–131, Mar. 2009.

[96] M. Michel and V. Stanford, “Synchronizing multimodal data streams acquired using commodity hardware,” in *Proceedings of the 4th ACM international workshop on Video surveillance and sensor networks*, 2006, pp. 3–8.

[97] J. Wagner, F. Lingenfelser, E. André, D. Mazzei, A. Tognetti, A. Lanatà, D. De Rossi, A. Betella, R. Zucca, P. Omedas, and P. F. M. J. Verschure, “A Sensing Architecture for Empathetic Data Systems,” in *Proceedings of the 4th Augmented Human International Conference*, New York, NY, USA, 2013, pp. 96–99.

[98] C. Neuper, M. Wörtz, and G. Pfurtscheller, “ERD/ERS patterns reflecting sensorimotor activation and deactivation,” *Prog. Brain Res.*, vol. 159, pp. 211–222, 2006.

[99] C. M. Krause, A. Heikki Lang, M. Laine, M. Kuusisto, and B. Pörn, “Event-related. EEG desynchronization and synchronization during an auditory memory task,” *Electroencephalogr. Clin. Neurophysiol.*, vol. 98, no. 4, pp. 319–326, Apr. 1996.

[100] M. Chen, “A Low-latency Lip-synchronized Videoconferencing System,” in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, New York, NY, USA, 2003, pp. 465–471.

[101] I. Kouvelas, V. Hardman, and A. Watson, “Lip synchronisation for use over the Internet: analysis and implementation,” in *Global Telecommunications Conference, 1996. GLOBECOM ’96. ’Communications: The Key to Global Prosperity*, 1996, vol. 2, pp. 893–898 vol.2.

[102] R. Steinmetz, “Human perception of jitter and media synchronization,” *IEEE J. Sel. Areas Commun.*, vol. 14, no. 1, pp. 61–72, Jan. 1996.

[103] J. E. Elson, “Time synchronization in wireless sensor networks,” University of California, Los Angeles, 2003.

[104] J. Elson and K. Römer, “Wireless sensor networks: A new regime for time synchronization,” *ACM SIGCOMM Comput. Commun. Rev.*, vol. 33, no. 1, pp. 149–154, 2003.

[105] J. Elson and D. Estrin, “Time Synchronization for Wireless Sensor Networks,” in *Proceedings of the 15th International Parallel & Distributed Processing Symposium*, Washington, DC, USA, 2001, p. 186–.

[106] M. Huber, M. Schlegel, and G. Klinker, “Temporal calibration in multisensor tracking setups,” in *Mixed and Augmented Reality, 2009. ISMAR 2009. 8th IEEE International Symposium on*, 2009, pp. 195–196.

[107] Z. Yu and Y. Nakamura, “Smart meeting systems: A survey of state-of-the-art and open issues,” *ACM Comput. Surv. CSUR*, vol. 42, no. 2, p. 8, 2010.

[108] J. C. P. Chan, H. Leung, J. K. T. Tang, and T. Komura, “A Virtual Reality Dance Training System Using Motion Capture Technology,” *Learn. Technol. IEEE Trans. On*, vol. 4, no. 2, pp. 187 –195, Jun. 2011.

[109] C. Zrenner, D. Eytan, A. Wallach, P. Thier, and S. Marom, “A generic framework for real-time multi-channel neuronal signal analysis, telemetry control, and sub-millisecond latency feedback generation,” *Front. Neuroprosthetics*, vol. 4, p. 173, 2010.

[110] C. Just, C. Cruz-Neira, and A. Baker, “Performance measurement capabilities of vr juggler: real-time monitoring of immersive applications,” in *The 4th international immersive projection technology workshop*, 2000.

[111] R. Pausch, T. Crea, and M. Conway, “A Literature Survey for Virtual Environments: Military Flight Simulator Visual Systems and Simulator Sickness,” *Presence Teleoper Virtual Env.*, vol. 1, no. 3, pp. 344–363, Jul. 1992.

[112] E. M. Kolasinski, “Simulator Sickness in Virtual Environments,” DTIC Document, Final technical rept. Aug 93-Dec 9 ADA295861, May 1995.

[113] “AndroidLatency — PD Community Site.” [Online]. Available: http://puredata.info/docs/AndroidLatency/. [Accessed: 20-Nov-2013].

[114] A. Camurri, P. Coletta, G. Varni, and S. Ghisio, “Developing Multimodal Interactive Systems with EyesWeb XMI,” in *Proceedings of the 7th International Conference on New Interfaces for Musical Expression*, New York, NY, USA, 2007, pp. 305–308.

[115] S. Kumar, “Challenges for Ubiquitous Computing,” in *Fifth International Conference on Networking and Services, 2009. ICNS ’09*, 2009, pp. 526 –535.

[116] S. Baruah, V. Bonifaci, G. D’Angelo, H. Li, A. Marchetti-Spaccamela, N. Megow, and L. Stougie, “Scheduling Real-Time Mixed-Criticality Jobs,” *Comput. IEEE Trans. On*, vol. 61, no. 8, pp. 1140–1152, 2012.

[117] M. Neukirchner, S. Quinton, R. Ernst, and K. Lampka, “Multi-mode monitoring for mixed-criticality real-time systems,” in *Hardware/Software Codesign and System Synthesis (CODES+ISSS), 2013 International Conference on*, 2013, pp. 1–10.

[118] “Ergoneers GmbH: Ergonomie, Blickerfassung, Blickerfassungssysteme - Produkte: Dikablis & D-Lab - Data Analysis.” [Online]. Available: http://www.ergoneers.com/en/products/dlab-dikablis/dataanalysis.html. [Accessed: 11-Mar-2012].

[119] “Dikablis & D-Lab - Data Analysis.” [Online]. Available: http://www.ergoneers.com/en/products/dlab-dikablis/dataanalysis.html. [Accessed: 11-Mar-2012].

[120] “Mind Media B.V. Information about BioTrace+ Software.” [Online]. Available: http://www.mindmedia.nl/english/biotrace.php. [Accessed: 11-Mar-2012].

[121] J. Wagner, E. Andre, and F. Jung, “Smart sensor integration: A framework for multimodal emotion recognition in real-time,” in *3rd International Conference on Affective Computing and Intelligent Interaction and Workshops, 2009. ACII 2009*, 2009, pp. 1–8.

[122] J. Wagner, F. Lingenfelser, and E. André, “The Social Signal Interpretation Framework (SSI) for Real Time Signal Processing and Recognition,” in *Twelfth Annual Conference of the International Speech Communication Association*, 2011.

[123] A. Benoit, L. Bonnaud, A. Caplier, F. Jourde, L. Nigay, M. Serrano, I. Damousis, D. Tzovaras, and J.-Y. L. Lawson, “Multimodal signal processing and interaction for a driving simulator: component-based architecture,” *J. Multimodal User Interfaces*, vol. 1, no. 1, pp. 49–58, 2007.

[124] A. Camurri, P. Coletta, A. Massari, B. Mazzarino, M. Peri, M. Ricchetti, A. Ricci, and G. Volpe, “Toward real-time multimodal processing: EyesWeb 4.0,” presented at the AISB 2004 Convention: Motion, Emotion and Cognition, 2004.

[125] A. Camurri, P. Coletta, M. Demurtas, M. Peri, A. Ricci, R. Sagoleo, M. Simonetti, G. Varni, and G. Volpe, “An Open Platform for Multimodal Data Streams Processing.” .

[126] “Max is powerful software. Cycling 74.” [Online]. Available: http://cycling74.com/products/max/. [Accessed: 20-Nov-2013].

[127] T. Place and T. Lossius, “Jamoma: A modular standard for structuring patches in Max,” in *Proceedings, International Computer Music Conference, New Orleans*, 2006.

[128] “Pure Data — PD Community Site.” [Online]. Available: http://puredata.info/. [Accessed: 15-Nov-2013].

[129] M. Puckette, “Pure data: Recent progress,” in *Proceedings of the Third Intercollege Computer Music Festival*, 1997, pp. 1–4.

[130] M. Puckette, “Pure Data: another integrated computer music environment,” in *in Proceedings, International Computer Music Conference*, 1996, pp. 37–41.

[131] J. H. Kim, D. V. Gunn, E. Schuh, B. Phillips, R. J. Pagulayan, and D. Wixon, “Tracking real-time user experience (TRUE): a comprehensive instrumentation solution for complex systems,” in *Proceedings of the twenty-sixth annual SIGCHI conference on Human Factors in Computing Systems*, 2008, pp. 443–452.

[132] W. Steptoe and A. Steed, “Multimodal Data Capture and Analysis of Interaction in Immersive Collaborative Virtual Environments,” *Presence Teleoperators Virtual Environ.*, vol. 21, no. 4, pp. 388–405, Nov. 2012.

[133] A. Bierbaum, C. Just, P. Hartling, K. Meinert, A. Baker, and C. Cruz-Neira, “VR Juggler: a virtual platform for virtual reality application development,” in *IEEE Virtual Reality, 2001. Proceedings*, 2001, pp. 89–96.

[134] A. Bierbaum and C. Cruz-Neira, “Run-time reconfiguration in VR Juggler,” in *4th Immersive Projection Technology Workshop*, 2000.

[135] D. Heger, F. Putze, C. Amma, M. Wand, I. Plotkin, T. Wielatt, and T. Schultz, “BiosignalsStudio: a flexible framework for biosignal capturing and processing,” *KI 2010 Adv. Artif. Intell.*, pp. 33–39, 2010.

[136] W. Li, Y.-H. Lee, W.-T. Tsai, J. Xu, Y.-S. Son, J.-H. Park, and K.-D. Moon, “Service-oriented smart home applications: composition, code generation, deployment, and execution,” *Serv. Oriented Comput. Appl.*, vol. 6, no. 1, pp. 65–79, Mar. 2012.

[137] M. Rojc and I. Mlakar, “Finite-state machine based distributed framework DATA for intelligent ambience systems,” in *Proceedings of the 8th WSEAS International Conference on Computational intelligence, man-machine systems and cybernetics*, 2009, pp. 80–85.

[138] “W3C Multimodal Architecture and Interfaces,” *Wikipedia, the free encyclopedia*. 06-Feb-2013.

[139] M. Morisio, D. Romano, and I. Stamelos, “Quality, productivity, and learning in framework-based development: an exploratory case study,” *IEEE Trans. Softw. Eng.*, vol. 28, no. 9, pp. 876–888, 2002.

[140] H. Mili, M. Fayad, D. Brugali, D. Hamu, and D. Dori, “Enterprise frameworks: issues and research directions,” *Softw. Pract. Exp.*, vol. 32, no. 8, pp. 801–831, 2002.

[141] A. Valerio, G. Succi, and M. Fenaroli, “Domain analysis and framework-based software development,” *SIGAPP Appl Comput Rev*, vol. 5, no. 2, pp. 4–15, Sep. 1997.

[142] W. Codenie, K. De Hondt, P. Steyaert, and A. Vercammen, “From custom applications to domain-specific frameworks,” *Commun ACM*, vol. 40, no. 10, pp. 70–77, Oct. 1997.

[143] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design patterns: elements of reusable object-oriented software*. Boston [u.a.: Addison-Wesley, 1998.

[144] J. van Gurp and J. Bosch, “Design, implementation and evolution of object oriented frameworks: concepts and guidelines,” *Softw. Pract. Exp.*, vol. 31, no. 3, pp. 277–300, Mar. 2001.

[145] M. Mattsson, “Evolution and composition of object-oriented frameworks,” University of Karlskrona/Ronneby. Sweden, 2000.

[146] D. Roberts and R. Johnson, “Evolving Frameworks: A Pattern Language for Developing Object-Oriented Frameworks,” *Proc. Third Conf. Pattern Lang. Program.*, 1996.

[147] H. A. Schmid, “Systematic framework design by generalization,” *Commun ACM*, vol. 40, no. 10, pp. 48–51, Oct. 1997.

[148] J. A. Kim, “How to develop and to reuse the UniPDM framework,” *Softw Pr. Exper*, vol. 32, no. 8, pp. 737–754, Jul. 2002.

[149] K. Cwalina and B. Abrams, *Framework Design Guidelines: Conventions, Idioms, and Patterns for Reusable .NET Libraries*. Pearson Education, 2008.

[150] D. Wilson and S. Wilson, “Writing frameworks-capturing your expertise about a problem domain,” *Tutor. Notes OOPSLA’93*, 1993.

[151] R. E. Johnson and W. F. Opdyke, “Refactoring and aggregation,” in *Object Technologies for Advanced Software*, S. Nishio and A. Yonezawa, Eds. Springer Berlin Heidelberg, 1993, pp. 264–278.

[152] C. E. Cummings, “Simulation and synthesis techniques for asynchronous FIFO design,” in *SNUG 2002 (Synopsys Users Group Conference, San Jose, CA, 2002) User Papers*, 2002.

[153] G. Ramesh, V. S. Kumar, and K. J. Reddy, “Asynchronous FIFO Design with Gray code Pointer for High Speed AMBA AHB Compliant Memory controller.”

[154] A. Williams, *C++ Concurrency in Action: Pratical Multithreading*. Shelter Island: Manning, 2011.

[155] J. Elson, L. Girod, and D. Estrin, “Fine-grained network time synchronization using reference broadcasts,” *ACM SIGOPS Oper. Syst. Rev.*, vol. 36, no. SI, pp. 147–163, 2002.

[156] A. Bechara, H. Damasio, and A. R. Damasio, “Emotion, Decision Making and the Orbitofrontal Cortex,” *Cereb. Cortex*, vol. 10, no. 3, pp. 295–307, Mar. 2000.

[157] A. P. Conway and W. J. Ion, “Enhancing the design dialogue: an architecture to document engineering design activities,” *J. Eng. Des.*, vol. 24, no. 2, pp. 140–164, 2013.

[158] S. Szykman, R. D. Sriram, C. Bochenek, J. W. Racz, and J. Senfaute, *Design Repositories: Next-Generation Engineering Design Databases*. 2000.

[159] Y. Jin and Y. Ishino, “DAKA: design activity knowledge acquisition through data-mining,” *Int. J. Prod. Res.*, vol. 44, no. 14, pp. 2813–2837, Jul. 2006.

[160] F. M. Shipman and R. J. McCall, “Integrating different perspectives on design rationale: Supporting the emergence of design rationale from design communication,” *Artif. Intell. Eng. Des. Anal. Manuf.*, vol. 11, no. 02, pp. 141–154, 1997.

[161] D. R. Campbell, S. J. Culley, C. A. McMahon, and P. Coleman, “A Methodology for Profiling Computer Based Design Activities,” *ICED 05 15th Int. Conf. Eng. Des. Eng. Des. Glob. Econ.*, p. 318, 2005.

[162] D. R. Campbell, S. J. Culley, C. A. McMahon, and F. Sellini, “An approach for the capture of context-dependent document relationships extracted from Bayesian analysis of users’ interactions with information,” *Inf. Retr.*, vol. 10, no. 2, pp. 115–141, Oct. 2006.

[163] G. Huet, “Design transaction monitoring : understanding design reviews for extended knowledge capture,” Ph.D., University of Bath, 2006.

[164] A. Conway, A. Wodehouse, W. Ion, and N. Juster, “A study of information & knowledge generated during engineering design meetings,” in *International Conference on Engineering Design (ICED)*, 2007.

[165] H. J. Rea, I. K. Howley, J. R. Corney, J. M. Ritchie, R. Sung, and C. Salamon, “CBBC BAMZOOKi ™ as a Tool for Engineering Design Research,” in *Proceedings of Learning with Games (LG2007)*, Sophia Antipolis, France, 2007, pp. 585–594.

[166] “IDEF0,” *Wikipedia, the free encyclopedia*. 09-Apr-2013.

[167] R. H. Bracewell, S. Ahmed, and K. M. Wallace, “DRed and design folders: a way of capturing, storing and passing on - knowledge generated during design projects,” in *Proceedings of the 2004 ASME International Design Engineering Technical Conferences (DETC’04); Vol. 1*, Salt Lake City, Utah, USA, 2004, pp. 235–246.

[168] J. M. Ritchie, R. Sung, H. Rea, T. Lim, J. R. Corney, C. Salamon, and I. Howley, “Automated knowledge capture in 2D and 3D design environments,” in *2nd*

International Workshop Virtual Manufacturing VirMan 08 as part of the 5th INTUITION International Conference: Virtual Reality in Industry and Society: From Research to Application, 2008.

[169] R. C. W. Sung, J. M. Ritchie, T. Lim, and Z. Kosmadoudi, "Automated generation of engineering rationale, knowledge and intent representations during the product life cycle," *Virtual Real.*, vol. 16, no. 1, pp. 69–85, Aug. 2011.

[170] A. N. Dragunov, T. G. Dietterich, K. Johnsrude, M. McLaughlin, L. Li, and J. L. Herlocker, "TaskTracer: a desktop environment to support multi-tasking knowledge workers," in *Proceedings of the 10th international conference on Intelligent user interfaces*, 2005, pp. 75–82.

[171] D. M. Hilbert and D. F. Redmiles, "Extracting usability information from user interface events," *ACM Comput. Surv. CSUR*, vol. 32, no. 4, pp. 384–421, 2000.

[172] K. D. Fenstermacher and M. Ginsburg, "A lightweight framework for cross-application user monitoring," *Computer*, vol. 35, no. 3, pp. 51–59, 2002.

[173] V. Kaptelinin, "UMEA: translating interaction histories into project contexts," in *Proceedings of the SIGCHI conference on Human factors in computing systems*, 2003, pp. 353–360.

[174] R. Lokaiczyk, A. Faatz, A. Beckhaus, and M. Goertz, "Enhancing Just-in-Time E-Learning Through Machine Learning on Desktop Context Sensors," in *Modeling and Using Context*, vol. 4635, B. Kokinov, D. Richardson, T. Roth-Berghofer, and L. Vieu, Eds. Springer Berlin / Heidelberg, 2007, pp. 330–341.

[175] K. R. Müller, M. Tangermann, G. Dornhege, M. Krauledat, G. Curio, and B. Blankertz, "Machine learning for real-time single-trial EEG-analysis: From brain-computer interfacing to mental state monitoring," *J. Neurosci. Methods*, vol. 167, no. 1, pp. 82–90, 2008.

[176] D. Heger, F. Putze, and T. Schultz, "Online Workload Recognition from EEG Data during Cognitive Tests and Human-Machine Interaction," in *KI 2010: Advances in Artificial Intelligence*, vol. 6359, R. Dillmann, J. Beyerer, U. Hanebeck, and T. Schultz, Eds. Springer Berlin / Heidelberg, 2010, pp. 410–417.

[177] I. M. Rezazadeh, X. Wang, M. Firoozabadi, and M. R. Hashemi Golpayegani, "Using affective human–machine interface to increase the operation performance in virtual construction crane training system: A novel approach," *Autom. Constr.*, Nov. 2010.

[178] M. Soleymani, J. Lichtenauer, T. Pun, and M. Pantic, “A Multi-Modal Affective Database for Affect Recognition and Implicit Tagging,” *IEEE Trans. Affect. Comput.*, 2011.

[179] R. Sung, J. M. Ritchie, H. J. Rea, and J. Corney, “Automated design knowledge capture and representation in single-user CAD environments,” *J. Eng. Des.*, vol. 22, no. 7, pp. 487–503, Jul. 2011.

[180] Y. Liu, Z. Kosmadoudi, R. Sung, T. Lim, S. Louchart, and J. Ritchie, “Capture User Emotions during Computer- Aided Design,” in *Proceedings of IDMME - Virtual Concept 2010*, Bordeaux, France, 2010.

[181] X.-W. Wang, D. Nie, and B.-L. Lu, “EEG-Based Emotion Recognition Using Frequency Domain Features and Support Vector Machines,” in *Neural Information Processing*, B.-L. Lu, L. Zhang, and J. Kwok, Eds. Springer Berlin Heidelberg, 2011, pp. 734–743.

[182] C. van Nimwegen and A. J. Uyttendaele, “Unobtrusive physiological measures to adapt system behavior: The GSR mouse,” in *status: published*, Brighton, Sussex, UK, 2009.

[183] T. Partala and V. Surakka, “Pupil size variation as an indication of affective processing,” *Int. J. Hum.-Comput. Stud.*, vol. 59, no. 1–2, pp. 185–198, 2003.

[184] Y. Ayzenberg, J. Hernandez Rivera, and R. Picard, “FEEL: frequent EDA and event logging–a mobile social interaction stress monitoring system,” in *Proceedings of the 2012 ACM annual conference extended abstracts on Human Factors in Computing Systems Extended Abstracts*, 2012, pp. 2357–2362.

[185] J. Klein, Y. Moon, and R. W. Picard, “This computer responds to user frustration:: Theory, design, and results,” *Interact. Comput.*, vol. 14, no. 2, pp. 119–140, 2002.

[186] S. K. Chandrasegaran, K. Ramani, R. D. Sriram, I. Horváth, A. Bernard, R. F. Harik, and W. Gao, “The evolution, challenges, and future of knowledge representation in product design systems,” *Comput.-Aided Des.*, vol. 45, no. 2, pp. 204–228, Feb. 2013.

[187] M. Langheinrich, *Privacy in Ubiquitous Computing*. Chapman and Hall/CRC, 2013.

[188] J. I. Hong and J. A. Landay, “An architecture for privacy-sensitive ubiquitous computing,” in *Proceedings of the 2nd international conference on Mobile systems, applications, and services*, 2004, pp. 177–189.

[189] G. D. Nord, T. F. McCubbins, and J. H. Nord, “E-monitoring in the workplace: privacy, legislation, and surveillance software,” *Commun ACM*, vol. 49, no. 8, pp. 72–77, Aug. 2006.

[190] S. Miller and J. Weckert, “Privacy, the Workplace and the Internet,” *J. Bus. Ethics*, vol. 28, no. 3, pp. 255–265, Dec. 2000.

[191] R. C. Sung, J. M. Ritchie, T. Lim, Y. Liu, and Z. Kosmadoudi, “The Automated Generation of Engineering Knowledge Using A Digital Engineering Tool: An Industrial Evaluation Case Study,” *Int. J. Innov. Technol. Manag.*, vol. 09, no. 06, p. 1271001, Dec. 2012.

[192] F. Ameri and D. Dutta, “Product lifecycle management: closing the knowledge loops,” *Comput.-Aided Des. Appl.*, vol. 2, no. 5, pp. 577–590, 2005.

[193] A. Bernard, F. Laroche, and C. da Cunha, “Models and methods for knowledge formalisation in a PLM context,” in *actes de la conférence CMSM’2009*, 2009.

[194] G. L. Rocca, “Knowledge based engineering: Between AI and CAD. Review of a language based technology to support engineering design,” *Adv. Eng. Inform.*, vol. 26, no. 2, pp. 159–179, Apr. 2012.

[195] D. A. Mackall, *A Knowledge-based System Design/information Tool for Aircraft Flight Control Systems*. 1989.

[196] M. E. Porter, *Competitive Advantage: Creating and Sustaining Superior Performance*. Simon and Schuster, 2008.

[197] W. Wood, M. Yang, and M. Cutkosky, “Design Information Retrieval: Improving Access to the Informal Side of Design,” in *Proceedings of the ASME Design Engineering Technical Conferences*, 1998.

[198] J. Glesinger, “The not-so-hidden cost of lost knowledge,” *Energy Institute Future Skills*, p. 11, 2010.

[199] M. Grossman and S. Bates, “Knowledge capture within the biopharmaceutical clinical trials environment,” *VINE*, vol. 38, no. 1, pp. 118–132, Apr. 2008.

[200] W. J. C. Verhagen, P. Bermell-Garcia, R. E. C. van Dijk, and R. Curran, “A critical review of Knowledge-Based Engineering: An identification of research challenges,” *Adv. Eng. Inform.*, vol. 26, no. 1, pp. 5–15, Jan. 2012.

[201] K. Lockwood, “Using Analogy to Model Spatial Language Use and Multimodal Knowledge Capture,” Doctor of Philosophy, Northwestern University, Evanston, Illinois, 2009.

[202] David Baxter, X. Gao, and R. Roy, “Design Process Knowledge Reuse Challenges and Issues,” *Comput.-Aided Des. Appl.*, vol. 5, no. 6, 2008.

[203] J. M. Ritchie, R. G. Dewar, G. Robinson, J. E. L. Simmons, and F. M. Ng, “The role of non-intrusive operator logging to support the analysis and generation of product engineering data using immersive VR,” *Virtual Phys. Prototyp.*, vol. 1, no. 2, pp. 117–134, 2006.

[204] G. Robinson, J. M. Ritchie, P. N. Day, and R. G. Dewar, “System design and user evaluation of Co-Star: An immersive stereoscopic system for cable harness design,” *Comput Aided Des*, vol. 39, no. 4, pp. 245–257, Apr. 2007.

[205] R. C. W. Sung, J. M. Ritchie, G. Robinson, P. N. Day, J. R. Corney, and T. Lim, “Automated design process modelling and analysis using immersive virtual reality,” *Comput.-Aided Des.*, vol. 41, no. 12, pp. 1082–1094, Dec. 2009.

[206] P. L. Francis, *Best Practices: High Levels of Knowledge at Key Points Differentiate Commercial Shipbuilding from Navy Shipbuilding*. DIANE Publishing, 2009.

[207] “NVivo 10 research software for analysis and insight.” [Online]. Available: http://www.qsrinternational.com/products_nvivo.aspx. [Accessed: 11-Aug-2013].

[208] “Alcove9 - Unified Information Access (UIA) search engine solutions.” [Online]. Available: http://www.alcove9.com/. [Accessed: 11-Aug-2013].

[209] “AVEVA | Marine Design & Engineering Software - AVEVA Marine.” [Online]. Available: http://www.aveva.com/en/Products_and_Services/AVEVA_for_Marine/AVEVA_Marine.aspx. [Accessed: 11-Aug-2013].

[210] A. Bragdon, R. DeLine, K. Hinckley, and M. R. Morris, “Code space: touch + air gesture hybrid interactions for supporting developer meetings,” in *Proceedings of the ACM International Conference on Interactive Tabletops and Surfaces*, New York, NY, USA, 2011, pp. 212–221.

[211] X. Wang and P. S. Dunston, “Comparative Effectiveness of Mixed Reality-Based Virtual Environments in Collaborative Design,” *IEEE Trans. Syst. Man Cybern. Part C Appl. Rev.*, vol. 41, no. 3, pp. 284–296, 2011.

[212] J. Verlinden, I. Horváth, and T.-J. Nam, “Recording augmented reality experiences to capture design reviews,” *Int. J. Interact. Des. Manuf. IJIDeM*, vol. 3, no. 3, pp. 189–200, Aug. 2009.

[213] R. Owen and I. Horváth, “Towards Product-Related Knowledge Asset Warehousing in Enterprises,” in *in Proceedings of the Fourth International Symposium on Tools and Methods of Competitive Engineering*, 2002.

[214] K. Wallace and L. Blessing, *Engineering Design: A Systematic Approach*. Springer, 2007.

[215] “Qt Project.” [Online]. Available: http://qt-project.org/. [Accessed: 19-Sep-2013].

[216] “VTK - The Visualization Toolkit.” [Online]. Available: http://www.vtk.org/. [Accessed: 19-Sep-2013].

[217] J. Brooke, “SUS: A quick and dirty usability scale,” in *Usability evaluation in industry*, vol. 189, P. Jordan, B. Weerdmeester, A. Thomas, and I. Mclelland, Eds. Taylor and Francis, 1996, p. 194.

[218] Brooke, John, “SUS: A Retrospective,” *J. Usability Stud.*, vol. 8, no. 2, pp. 29–40, Feb. 2013.

[219] K. Dalkir, *Knowledge management in theory and practice*. Cambridge, Mass.: MIT Press, 2011.

[220] T. H. Davenport and L. Prusak, *Information Ecology: Mastering the Information and Knowledge Environment*, 1st ed. Oxford University Press, 1997.

[221] M. Steyvers and T. Griffiths, “Probabilistic topic models,” *Handb. Latent Semantic Anal.*, vol. 427, no. 7, pp. 424–440, 2007.

[222] T. K. Landauer, D. S. McNamara, S. Dennis, and W. Kintsch, *Handbook of Latent Semantic Analysis*. Psychology Press, 2013.

[223] D. Mimno, “Computational historiography: Data mining in a century of classics journals,” *J Comput Cult Herit*, vol. 5, no. 1, pp. 3:1–3:19, Apr. 2012.

[224] “HWR - Heriot Watt’s Entry in Formula Student.” [Online]. Available: http://www.mec.hw.ac.uk/formula-student/. [Accessed: 25-Jan-2013].

[225] D. Sasse, “A framework for psychophysiological data acquisition in digital games,” Blekinge Institute of Technology, 2008.

[226] A. Nijholt, D. P. O. Bos, and B. Reuderink, “Turning shortcomings into challenges: Brain–computer interfaces for games,” *Entertain. Comput.*, vol. 1, no. 2, pp. 85–94, 2009.

[227] J. M. Kivikangas, L. Nacke, and N. Ravaja, “Developing a triangulation system for digital game events, observational video, and psychophysiological data to study emotional responses to a virtual character,” *Entertain. Comput.*, vol. 2, no. 1, pp. 11 – 16, 2011.

[228] G. G. Molina, T. Tsoneva, and A. Nijholt, “Emotional brain-computer interfaces,” in *3rd International Conference on Affective Computing and Intelligent Interaction and Workshops, 2009. ACII 2009*, 2009, pp. 1 –9.

[229] C.-T. Lin, I.-F. Chung, L.-W. Ko, Y.-C. Chen, S.-F. Liang, and J.-R. Duann, "EEG-Based Assessment of Driver Cognitive Responses in a Dynamic Virtual-Reality Driving Environment," *IEEE Trans. Biomed. Eng.*, vol. 54, no. 7, pp. 1349 –1352, Jul. 2007.

[230] C.-T. Lin, S.-A. Chen, L.-W. Ko, and Y.-K. Wang, "EEG-based brain dynamics of driving distraction," in *The 2011 International Joint Conference on Neural Networks (IJCNN)*, 2011, pp. 1497 –1500.

[231] S. Lei, "Driver Mental States Monitoring Based on Brain Signals," PhD Thesis, Berlin Institute of Technology, 2011.

[232] A. D. Ohlhauser, S. Milloy, and J. K. Caird, "Driver responses to motorcycle and lead vehicle braking events: The effects of motorcycling experience and novice versus experienced drivers," *Transp. Res. Part F Traffic Psychol. Behav.*, vol. 14, no. 6, pp. 472–483, Nov. 2011.

[233] S. Haufe, M. S. Treder, M. F. Gugler, M. Sagebaum, G. Curio, and B. Blankertz, "EEG potentials predict upcoming emergency brakings during simulated driving," *J. Neural Eng.*, vol. 8, no. 5, 2011.

[234] M. de Haan, Ed., *Infant EEG and Event-Related Potentials*, 1st ed. Psychology Press, 2007.

[235] M. M. Doran and J. E. Hoffman, "The role of visual attention in multiple object tracking: Evidence from ERPs," *Atten. Percept. Psychophys.*, vol. 72, no. 1, pp. 33–52, 2010.

[236] J. L. Florens, A. Luciani, C. Cadoz, and N. Castagné, "ERGOS: multi-degrees of freedom and versatile force-feedback panoply," in *Proceedings of EuroHaptics*, 2004, pp. 356–360.

[237] A. Luciani, J.-L. Florens, and N. Castagne, "From action to sound: a challenging perspective for haptics," in *Eurohaptics Conference, 2005 and Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems, 2005. World Haptics 2005. First Joint*, 2005, pp. 592 – 595.

[238] J. L. Crowley and J. Martin, "Visual Processes for Tracking and Recognition of Hand Gestures," in *In proceedings of Workshop on Perceptual User Interfaces (PUI'97)*, In Workshop on Perceptual User Interfaces (PUI'97), 1997.

[239] "HANS device," *Wikipedia, the free encyclopedia*. 16-Jan-2013.

[240] “British Motor Racing Circuits - Silverstone Circuit Maps - Stowe Circuit.” [Online]. Available: http://www.silverstone.co.uk/track/circuit-maps/stowe-circuit/. [Accessed: 25-Jan-2013].

[241] J. N. Demos, *Getting Started with Neurofeedback*. W. W. Norton & Company, 2005.

[242] R. Grega, “Dealing with Noise in EEG Recording and Data Analysis,” *Inform. Medica Slov.*, vol. 15, no. 1, pp. 18–25, Sep. 2010.

[243] J. Mateo, A. M. Torres, C. Soria, and J. L. Santos, “A method for removing noise from continuous brain signal recordings,” *Comput. Electr. Eng.*, vol. 39, no. 5, pp. 1561–1570, Jul. 2013.

[244] F. Lotte, J. Fujisawa, H. Touyama, R. Ito, M. Hirose, and A. Lécuyer, “Towards ambulatory brain-computer interfaces: a pilot study with P300 signals,” in *Proceedings of the International Conference on Advances in Computer Enterntainment Technology*, 2009, pp. 336–339.

[245] F. Hutzler, M. Braun, M. L. . Vő, V. Engl, M. Hofmann, M. Dambacher, H. Leder, and A. M. Jacobs, “Welcome to the real world: Validating fixation-related brain potentials for ecologically valid settings,” *Brain Res.*, vol. 1172, pp. 124–129, 2007.

[246] Y. Agam and R. Sekuler, “Interactions between working memory and visual perception: An ERP/EEG study,” *Neuroimage*, vol. 36, no. 3, pp. 933–942, 2007.

[247] R. Scheeringa, K. M. Petersson, A. Kleinschmidt, O. Jensen, and M. C. M. Bastiaansen, “EEG α power modulation of fMRI resting-state connectivity,” *Brain Connect.*, vol. 2, no. 5, pp. 254–264, 2012.

[248] T. Mulholland, “The concept of attention and the EEG alpha-rhythm,” *Electroencephalogr. Clin. Neurophysiol.*, vol. 24, no. 2, p. 188, Feb. 1968.

[249] W. J. Ray and H. W. Cole, “EEG alpha activity reflects attentional demands, and beta activity reflects emotional and cognitive processes,” *Science*, vol. 228, no. 4700, pp. 750–752, May 1985.

[250] W. Klimesch, M. Doppelmayr, H. Russegger, T. Pachinger, and J. Schwaiger, “Induced alpha band power changes in the human EEG and attention,” *Neurosci. Lett.*, vol. 244, no. 2, pp. 73–76, Mar. 1998.

[251] S. Palva and J. M. Palva, “New vistas for α-frequency band oscillations,” *Trends Neurosci.*, vol. 30, no. 4, pp. 150–158, Apr. 2007.

[252] C. J. Mann, “Observational research methods. Research design II: cohort, cross sectional, and case-control studies,” *Emerg. Med. J. EMJ*, vol. 20, no. 1, pp. 54–60, Jan. 2003.

[253] G. Pfurtscheller and F. H. Lopes da Silva, “Event-related EEG/MEG synchronization and desynchronization: basic principles,” *Clin. Neurophysiol.*, vol. 110, no. 11, pp. 1842–1857, Nov. 1999.

[254] L. Leocani, C. Toro, P. Manganotti, P. Zhuang, and M. Hallett, “Event-related coherence and event-related desynchronization/synchronization in the 10 Hz and 20 Hz EEG during self-paced movements,” *Electroencephalogr. Clin. Neurophysiol. Potentials Sect.*, vol. 104, no. 3, pp. 199–206, May 1997.

[255] J. Baumeister, T. Barthel, K. R. Geiss, and M. Weiss, “Influence of phosphatidylserine on cognitive performance and cortical activity after induced stress,” *Nutr. Neurosci.*, vol. 11, no. 3, pp. 103–110, Jun. 2008.

[256] Elisabeth V. C. Friedrich, Neil Suttie, Aparajithan Sivanathan, Theodore Lim, Sandy Louchart, and Jaime A. Pineda, “A Neurofeedback protocol using a sophisticated interactive video game to improve social responsiveness in children with ASD,” in *Society for Neuroscience, 2013*, New Orleans, LA, 2013, p. 181.15/AAA24.

[257] E. V. C. Friedrich, N. Suttie, A. Sivanathan, T. Lim, S. Louchart, and J. A. Pineda, “Brain-computer interface game applications for combined neurofeedback and biofeedback treatment for children on the autism spectrum,” *Front. Neuroengineering*, vol. 7, no. 21, 2014.

[258] C. E. Rice, “The changing prevalence of the autism spectrum disorders,” *Am. Fam. Physician*, vol. 83, no. 5, pp. 515–520, 2011.

[259] R. L. Hansen, S. Ozonoff, P. Krakowiak, K. Angkustsiri, C. Jones, L. J. Deprey, D.-N. Le, L. A. Croen, and I. Hertz-Picciotto, “Regression in Autism: Prevalence and Associated Factors in the CHARGE Study,” *Ambul. Pediatr.*, vol. 8, no. 1, pp. 25 – 31, 2008.

[260] A. P. Association, *Diagnostic and Statistical Manual of Mental Disorders, Fourth Edition: DSM-IV-TR®*. American Psychiatric Association, 2000.

[261] L. Thompson, M. Thompson, and A. Reid, “Functional Neuroanatomy and the Rationale for Using EEG Biofeedback for Clients with Asperger’s Syndrome,” *Appl. Psychophysiol. Biofeedback*, vol. 35, no. 1, pp. 39–61, Mar. 2010.

[262] R. Coben and T. E. Myers, “The Relative Efficacy of Connectivity Guided and Symptom Based EEG Biofeedback for Autistic Disorders,” *Appl. Psychophysiol. Biofeedback*, vol. 35, no. 1, pp. 13–23, Mar. 2010.

[263] S. W. Porges, “The polyvagal perspective,” *Biol. Psychol.*, vol. 74, no. 2, pp. 116–143, Feb. 2007.

[264] “Biofeedback Device: ProComp Infiniti Encoder - EEG, EKG, EMG, skin conductance, heart rate, blood volume pulse, respiration, goniometry, force, and voltage input.” [Online]. Available: http://www.thoughttechnology.com/proinf.htm. [Accessed: 28-Nov-2013].

[265] “Biofeedback Products: EEG, EMG, EKG, HRV, Muscle Rehabilitation, Continence Software.” [Online]. Available: http://www.thoughttechnology.com/software.htm. [Accessed: 28-Nov-2013].

[266] “C/C++ Compiler ARM and Cortex - IAR Embedded Workbench for ARM - IAR.” [Online]. Available: http://www.iar.com/Products/IAR-Embedded-Workbench/ARM/. [Accessed: 28-Nov-2013].

[267] M. E. Fayad, *Building application frameworks: object-oriented foundations of framework design*. New York, N.Y: Wiley, 1999.

[268] J. Bansiya and C. G. Davis, “A hierarchical model for object-oriented design quality assessment,” *IEEE Trans. Softw. Eng.*, vol. 28, no. 1, pp. 4–17, 2002.

[269] “MFC Reference.” [Online]. Available: http://msdn.microsoft.com/en-us/library/d06h2x6e(VS.90).aspx. [Accessed: 18-Oct-2013].

[270] “SourceForge.net: owlnext.” [Online]. Available: http://sourceforge.net/apps/mediawiki/owlnext/index.php?title=Main_Page. [Accessed: 18-Oct-2013].

[271] T. J. McCabe, “A Complexity Measure,” *IEEE Trans. Softw. Eng.*, vol. SE-2, no. 4, pp. 308–320, 1976.

[272] “Code Metrics Values.” [Online]. Available: http://msdn.microsoft.com/en-us/library/bb385914(v=vs.110).aspx. [Accessed: 19-Oct-2013].

[273] S. Joslin, R. Brown, and P. Drennan, “The gameplay visualization manifesto: a framework for logging and visualization of online gameplay data,” *Comput. Entertain. CIE*, vol. 5, no. 3, p. 6, 2007.

[274] M. Clark, K. Cellucci, C. Berka, D. Levendowski, J. Trejo, A. Kruse, and R. Stevens, “Enhancing team performance using neurophysiologic synchronies in a virtual training environment,” *Found. Augment. Cogn. Dir. Future Adapt. Syst.*, pp. 143–152, 2011.

[275] P. MJ and T. D. C. Littlez, “A Temporal Reference Framework for Multimedia Synchronization.”

[276] P.-A. Avouac, P. Lalanda, and L. Nigay, “Service-oriented Autonomic Multimodal Interaction in a Pervasive Environment,” in *Proceedings of the 13th International Conference on Multimodal Interfaces*, New York, NY, USA, 2011, pp. 369–376.

[277] A. Sivanathan, T. Lim, S. Louchart, and J. Ritchie, “Temporal Synchronisation of Data Logging in Racing Gameplay,” *Procedia Comput. Sci.*, vol. 15, pp. 103–110, 2012.

[278] A. Sivanathan, T. Lim, J. Ritchie, R. Sung, Z. Kosmadoudi, and Y. Liu, “The application of ubiquitous multimodal synchronous data capture in CAD,” *Comput.-Aided Des.*

[279] J. Wagner, F. Lingenfelser, T. Baur, I. Damian, F. Kistler, and E. André, “The social signal interpretation (SSI) framework: multimodal signal processing and recognition in real-time,” in *Proceedings of the 21st ACM international conference on Multimedia*, 2013, pp. 831–834.

Appendixes

Appendix 1. Hardware implementation - schematics and printed circuit board layout

Attached Disk :\tickku\

Appendix 2. Framework components usability metrics calculations

Attached Disk :\framework matrix.xlsx